EP 25879 (5)

PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 : G06F 9/44 — A1

(11) International Publication Number: **WO 00/48073**

(43) International Publication Date: 17 August 2000 (17.08.00)

(21) International Application Number: PCT/US00/03346

(22) International Filing Date: 9 February 2000 (09.02.00)

(30) Priority Data:
09/248,180 — 9 February 1999 (09.02.99) — US

(63) Related by Continuation (CON) or Continuation-in-Part (CIP) to Earlier Application
US — 09/248,180 (CON)
Filed on — 9 February 1999 (09.02.99)

(71) Applicant *(for all designated States except US)*: HEARME [US/US]; 665 Clyde Avenue, Mountain View, CA 94043 (US).

(72) Inventors; and

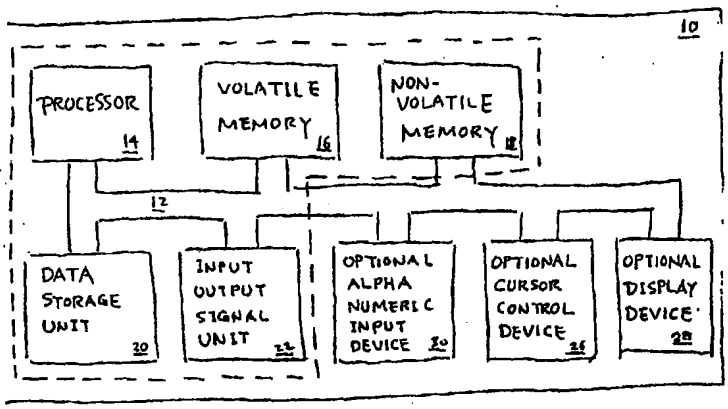
(75) Inventors/Applicants *(for US only)*: ROSKOWSKI, Steven, G. [US/US]; 1680 English Court, San Jose, CA 95129 (US). NA, Piaw [US/US]; 13376 Ronnie Way, Saratoga, CA 95070 (US).

(74) Agents: GALLENSON, Mavis, S. et al.; Ladas & Parry, Suite 2100, 5670 Wilshire Boulevard, Los Angeles, CA 90036–5679 (US).

(81) Designated States: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: METHOD AND APPARATUS FOR MANAGING ASSETS OF A CLIENT SIDE APPLICATION



(57) Abstract

The present invention simplifies the development of a client side application for a client–server application, wherein such a client side application typically runs on multiple clients having drastically different platform characteristics. To simplify the development process of a client side application, the present invention a) provides a programming model as a guide for code development, and b) supports a software component at each client machine's system level for asset management. The programming model guides the coding development of the client side application in a standardized and systematic manner. It also allows the developers to write codes that are platform–independent. Asset management is automated by a client system level software component (called the Asset Manager) running on each client's side. The Asset Manager of each client is tailored specifically for maximizing the capability of the client machine platform. Moreover, the Asset Manager of each client enables the development of client platform–independent code. In addition, the design of the Asset Manager via data type factories allows extensibility of media asset types; a new asset data type can easily be supported by simply adding a new data type factory to the Asset Manager without rewriting existing code. Finally, the Asset Manager uses a central location called the Asset Store to store assets shared by different client side applications, thus facilitating the speed of switching among these applications.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

METHOD AND APPARATUS FOR MANAGING ASSETS OF A CLIENT SIDE APPLICATION

FIELD OF THE INVENTION

The present invention relates to the client side development process of a client-server application. Specifically, the present invention offers a platform-independent programming model for efficiently developing and upgrading such client side application. Moreover, for each distinct client platform, the present invention manages the platform-specific programming logistics for delivering to the client machine large size data objects (assets) required by the client side application

BACKGROUND OF THE INVENTION

Client-server applications prevalent on the Internet today are mostly information based: applications serving information to the browser for display. But at the same time, exciting possibilities exist for developing more sophisticated client-server applications in at least several directions. One possible direction for the future of client-server applications is interaction oriented. Instead of browsing rather passively for information, the end users might want to be able interact with other users on-line (e.g. on-line gaming, on-line chatting). Another possible direction for the future of client-server applications is presentation oriented. The presentation of information would no longer be confined to the usual simple text and graphics data types. The focus of the application development would be on taking multi-media to a new level. Novel and effective formats of data types would continually be created to engage the users' interest and enhance the users' experience with richly presented information. Naturally, applications can also be both: offering high level of interactivity and presenting media rich information.

Commercially, developing more sophisticated client-server applications holds exciting promises. The user sessions for interactive applications are usually much longer than a browsing session. So interactive applications might offer an intriguing and effective medium for advertising. On the other hand, Web sites having rich media contents can attract more user visits. So richly presented applications also offers a potentially attractive medium for advertising.

However, developing these applications has been for the most part a slow process because of client platform proliferation. This is especially true for the client side application development process of a client-server application. So, before developers build interactive on-line applications, they must face and overcome a variety of problems in developing the client side application. What stands out among these problems is the proliferation of client machine platforms having drastically different characteristics such as CPU performance, connection bandwidth, persistent storage types.

Notwithstanding the existing variety of client platforms available now, the trend of new and innovative client platforms will surely continue unabated. Witness recently the introduction of a variety of set top boxes, network PC's, and handheld electronic devices. Thus, a urgent need arises for containing and handling the complexity of accommodating different client platforms in developing the client side applications. This is the hardware problem for developing client side applications.

Specifically, the proliferation of client platforms is a problem for developers of client side applications because no developer can control what an end user would use as a client machine. The developers are thus ignorant about the client platform characteristics on the client side. In writing any client side application, this ignorance would have to be compensated by extensive preparation on developers' part to ensure that the client side

application will work with any type (or at least most types) of client platforms in existence. Furthermore, given that each client side application can be designed for very different objectives, codes accommodating different client platforms in one client side application might not be viable for another type of client side application. A new set of codes might be necessary to accommodate different client platforms in a new on-line application. Consequently, developing client side applications is a painstakingly slow process.

Besides the hardware issue above for developing client side applications, the developers face another problem, this time coming from the software side. This problem is the proliferation of new asset types (data object types) involved in developing these client side applications. The assets required by a client side application are traditionally media assets such as image data objects, audio data objects, and text data objects. But the term "asset" can more broadly refer to any large size data type object needed to be delivered to the client in a speedy and precise manner, just like the media assets.

This proliferation of asset types is inevitable because in order to enhance an end user's on-line experience, developers constantly strive to offer a great variety of presentation modes. An end user running an interactive on-line application might want to listen to a certain audio files, then watch some animation. Each of these data type objects requires specific handling. By allowing multiple ways for an end user to experience information, an application enhances the experience of that information.

Given that an end user's on-line experience is enhanced by using interesting mixture of data object types (asset types), introducing new data object types enhances the application. But as new data object types become available for incorporating into the application, code rewriting can become quite extensive and prohibitive to implement.

How are these new data objects displayed to the end users? And for a large size data object, how will it be downloaded? These are critical questions that needed to be answered in codes by the developers. Thus, what is further needed is a way to handle the proliferation of data type objects being created for client side applications.

To make matter worse for the developers, when considering these software issues juxtaposed with the hardware issue of whether client machines of a particular platform can display or process the new data objects, developing a client side application can easily become a daunting task. In particular, in developing a client side application, developers face the intricate problem resulting from running various data objects on various client platforms. Handling the permutations of different data objects and client platforms entails a labor intensive task for developing client side applications. This is the coordination problem of simultaneously coping with both the hardware and the software proliferation problems.

For the rationale given above, in developing and running an client side application, what is needed is a way to overcome the hardware problem of client platform proliferation , the software problem of data type object proliferation, and the intricate coordination problem of running different data type objects on different client platform. The present invention solves both the hardware problem and the software problem involved in developing and running a client side application. The present invention also solves the intricate problem of running various data type objects on various client platforms.

SUMMARY OF THE INVENTION

The present invention simplifies the development process of the client side application for client-server applications, where the client side application is run on client machines of drastically different platforms. The present invention provides for the developers a powerful and useful infrastructure to quickly and precisely develop a client side application. In the process, the present invention ensures delivery of media rich, and/or rapidly refreshed images, while freeing the developer to focus on content aspect of the application. Specifically, to simplify the development process of a client side application, the present invention a) provides a platform independent programming model as a guide for code development, and b) supports a extensible software component (called the Asset Manager) at each client machine's system level for asset management. The platform-independent programming model solves problems of proliferation client machine platforms. The extensible design of the Asset Manager also solves problems of proliferating asset data types. Together, the programming model and the Asset Manager solves the coordination problem of mixing various different hardware platforms with various different software data types.

Programming Model according to the Present Invention:

An important aspect of the present invention is its programming model. This programming model, when followed, ensures that any asset being used by the client side application is being downloaded in an efficient manner, and is being delivered in an aggressive manner. In developing an client side application, the developers need to lay out just once the logistics of how various assets will be used by the application. This programming model standardizes the language, the API (Application Programming Interface), and the procedures for expressing the logistics of managing and delivering

assets needed to run an client side application. Through this programming model, the application developers can also prioritize all assets according to their urgency to running program by priority levels. Thus, the present invention's programming model offers a standardized way to express policies and logistics of utilizing assets according to these assets' priority levels. For example, a asset may be assigned a high priority level, a medium priority level, or a low priority level. Then, the developers can code the policy in a standardized way by coding that high priority assets must be downloaded before the medium priority assets, and so on.

Asset Manager according to the Present Invention:

At the core of the present invention is a software component called the *Asset Manager* that is running on each client machine. The Asset Manager is a system level software component. It provides an extensible architecture for handling and downloading new assets and providing them for use by other components in the client system. Advantageously, the design of the Asset Manager via data type factories allows extensibility of asset data types: a new asset data type can easily be supported by simply adding a new data type factory to the Asset Manager without rewriting existing code. Moreover, the present invention offers one type of Asset Manager per client machine platform by tailoring one Asset Manager specifically for each distinct client machine platform in order to realize each client machine's full potential. In other words, an Asset Manager can be viewed as a platform-specific system level software component of its associated client machine.

In a sense, before the complexity of dealing with multiple client platforms ever reach the application developers, the Asset Manager on each client machine is designed to

stop that client machine's distinct platform characteristics from ever becoming an issue for the developers. Better still, the Asset Manager only has to be designed once.

For example, for a client machine having high connection bandwidth and small dynamic memory size, the Asset Manager of this client machine would be designed to capitalize on the high connection bandwidth while compensating for small dynamic memory size by minimizing memory access. The emphasis here is on repeated download and diligent memory purging. In the reverse case, for a client machine having low connection bandwidth and large memory size, the Asset Manager of this client machine would be designed to capitalize on the large memory size by caching a great deal of downloaded information and minimizing download. The emphasis here is on caching, minimizing downloading. Moreover, as the client application runs, the Asset Manager keeps recently accessed assets in the Asset Store, a location in dynamic memory. Then, other client applications requiring the same assets can directly access these assets in the Asset Store. As a result, the use of Asset Store by the Asset Manager minimizes redundant asset download, and reduces the latency in switching between different client applications.

An important role of the Asset Manager is coordinating and finding assets wherever they are, pulling them together, ensuring the necessary assets are present for the client side application to run. Managing the localities of assets is a subtle process. Assets may be inter alia located on local persistent disk memory, CD-ROM, or the server machine, or etc. The Asset Manager keeps track of various asset images in various locations by using a global information text file typically called the Manifest. The Manifest is a list of assets together with their corresponding asset properties such as description, priority levels, quality levels. By working in concert with the information specified on the Manifest (by the developers or automatically generated), the Asset Manager ensures and delivers all the necessary assets with the optimal quality allowed by the client platform resources. The

Asset Manager will start the application before all the assets are available but continue background downloading the remaining assets as the application is running, all implemented without compromising the optimal quality allowed by the client platform resources.

Solving the Proliferation of Both Hardware Platforms and Software Data Objects:

One difficult task for developing a client side application is reconciling variety of client platforms and variety of data objects. The present invention, through the deployment of the Asset Manager running on each client machine, reconciles beautifully the hardware and software issues. In doing so, the present invention ensures that whatever the resources of each client machine are used fully and efficiently. Importantly, because of its flexibility to accommodate a wide range of platforms, the present invention enables platform-independence beyond that of "Mac. vs. PC".

In particular, one of the benefits of the present invention is that a developer needs only to concentrate on expressing the policies made up of priorities. The detailed implementation of carrying out these policies on a variety of client platforms is automatically done by the Asset Manager. Consequently, from the point of view of a developer, the proliferation of client machine platforms is no longer an obstacle. The programming model and the Asset Manager together ensure the getting of a necessary set of assets downloaded to the client machines at the necessary point in time in order for the application to proceed in a visually appealing and rational manner, and independent of client platforms. Moreover, from the point of view of an end user, the various differences of client platforms are transparent.

To summarize, the programming model offers a standardized way to organize, describe, and record assets. It supports multiple levels of quality for each asset and multiple levels of priority for assignment to each asset tagged with a certain quality level. In doing so, this programming model offers a standardized and convenient way to specify asset policies. The developers record the assets needed by the application on for example a text file called the Manifest. On each client machine, this Manifest is later used by the Asset Manager as the client application runs. The Asset Manager downloads the Manifest, and refers to it to implement the policies specified in the application's codes.

BRIEF DESCRIPTIONS OF THE DRAWINGS

The accompanying drawings, which are incorporated in and form a part of this specification, illustrate embodiments of the present invention and, together with the description, serve to explain the principles of the invention.

Figure 1 illustrates an exemplary computer system used as part of an asset management system in accordance with one embodiment of the present invention.

Figure 2 depicts a generic on-line setup for delivering information, where multiple clients are running browsers to access the information residing on a HTTP (Hyper Text Transfer Protocol) Web server across from the Internet.

Figure 3 depicts a client-server application more sophisticated than the one depicted in Figure 2. It is a chat application being run using the Internet. This illustrates an example of an interactive on-line application.

Figure 4 depicts a generic arrangement for running an interactive on-line application. These client machines are drawn with different sizes to indicate different platforms having different characteristics.

Figures 5A, and 5B are tables summarizing the possible client platform variations organized according to connection bandwidths, CPU speeds, and memory sizes. Figure 5 C gives a board summary of categories that can be used to distinguish various client platforms.

Figure 6 illustrates the preferred embodiment of the present invention, wherein various different client platforms are depicted. Some client machines are depicted as having local persistent storage's.

Figure 7 illustrates one embodiment of the present invention where one generic client machine has multiple local persistent storages. Each of these storage contains a different storage image of one asset.

Figure 8A depicts how assets listed in a Manifest are used in the programming model of the present invention. Figure 8B depicts the information organization of a generic Manifest.

Figure 9 depicts the internal design of the Asset Manager and illustrates how this design provides extensibility of asset data types. Figure 9 also illustrates the concept of the Asset Store and how it enables switching among client side applications.

Figure 10 is the flow chart associated with Figure 9.

Figure 11 is a flow chart of showing how the Asset Store facilitates application switching for the on-line game of Jeopardy.

DETAILED DESCRIPTION

A method and apparatus for management of assets used in running a client side application is described. In the following description, for purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be obvious, however, to one skilled in the art that the present invention may be practiced without these specific details. In other instances, well-known structures and devices are shown in block diagram form in order to avoid obscuring the present invention.

In the following detailed description of the present invention, some of the interchangeable key terms relating to the present invention are defined in the section below to resolve possible ambiguity and to facilitate future reference.

An *asset* is any data type object being used by a client side application. An asset traditionally refers to a media asset such as an image data object, an audio data object, or a text object. But an asset can also be a non-media asset that refers to any data object that needs to be delivered to the client machine for a client side application.

An *Asset Manager* is a system level software component running on a client machine. Each distinct client machine platform has its own type of Asset Manager.

Programming Model is inter alia a methodology of arranging and organizing code with a particular set of APIs for achieving certain specific purposes.

Quality Layers of an Asset refer to different possible versions of delivering an asset. In the case of an image asset, possible quality layers are black and white, and full color.

Priority Level of an Asset is the developer's assessment of the delivery order for an asset in relation to other assets.

Manifest is a text file containing a list of assets needed in running a client side application. This Manifest also records each asset's associated information such as the quality layer and the priority level to which each asset belongs.

Persistent Storage refers to any local persistent storage device of a client machine such as a harddrive or a CD-ROM.

Furthermore, unless specifically stated otherwise as apparent from the following discussions, it is appreciated that throughout the present invention, discussion utilizing terms such as "run", "download", "cache", "store", "switch", "fetch", "instantiate", or the like, refer to the actions and processes of a computer system, or similar electronic computing device. The computer system or similar electronic device manipulates and transforms data represented as physical (electronic) quantities within the computer system's registers and memories into other data similarly represented as physical quantities within the computer system memories into other data similarly represented as physical quantities within the computer system memories or registers or other such information storage, transmission, or display devices.

With reference to Figure 1, portions of the present invention are comprised of the computer-readable and computer executable instructions which reside, for example, in computer system 10 used as a part of a asset managing system in accordance with one embodiment of the present invention. It is appreciated that system 10 of Figure 1 is exemplary only and that the present invention can operate within a number of different computer systems including general-purpose computer systems, embedded computer systems, and stand-alone computer systems specially adapted for client side applications of client-server applications. Computer system 10 includes an address/data bus 12 for conveying digital information between the various components, a central processor unit (CPU) 14 for processing the digital information and instructions, a main memory 16 comprised of volatile random access memory (RAM) for storing the digital information and instructions, a non-volatile read only memory (ROM) 18 for storing information and instructions of a more permanent nature. In addition, computer system 10 may also include a data storage unit 20 (e.g., a magnetic, optical , floppy, or tape drive) for storing vast amounts of data, and an I/O interface 22 for interfacing with peripheral devices (e.g., computer network, modem, mass storage devices, etc.). It should be noted that the software component for performing the asset management process can be stored either in main memory 16, data storage unit 20, or in an external storage device. Devices which may be coupled to computer system 10 include a display device 28 for displaying information to a computer user, an alphanumeric input device 30 (e.g., a keyboard), and a cursor control device 26 (e.g., mouse, trackball, light pen, etc.) for inputting data, selections, updates, etc.

Generic Arrangement for Running an On-line Application:

Client-server applications prevalent using the Internet today are mostly information based. Figure 2 illustrates a generic arrangement for running such simple on-line

application over the Internet. Any of the three clients shown here (204, 206 and 208) can view the information on a Web page 270 by sending a request across the Internet 202 to a HTTP server 200 containing the Web page 270. This HTTP server 200 then sends the Web page 270 to the client that made the request. The on-line application in this case is not interactive in the sense that clients do not interact with each other.

Figure 3 illustrates the setup for running a chat room program, which is an interactive on-line application comprises a server application and a client application. This setup is similar to Figure 2 with the addition of an application server 300 hosting the chat server side application 370. The server 300 handles the logistics of various end users chatting with each other. In contrast with the on-line application of Figure 2, this chat application 370 is an *interactive* on-line application. Notice that the chat application comprises a server side application 370 and client side applications 374, 376, and 378 (i.e., one client side application of the chat application per one client machine).

The chat client applications 304, 306, and 308 run on clients 204, 206, and 208 respectively. Each of these three chat client applications 304, 306, and 308 works with browsers 274, 276, and 278 respectively to fetch assets 322 and 344, and to drive the client user graphic interface.

Figure 4 illustrates a generic setup similar to that of Figure 3 except that all three clients are depicted with different shapes for indicating that these three client machines belong to three *distinct* client platforms. Figure 4 provides a more realistic situation faced by a developer of the client side application for an interactive on-line applications. For a developer of client side interactive on-line applications, the objective is to deliver media rich applications. But the end users can be very far away, making connection bandwidth of the client machine a major consideration in delivering large sized asset such as a streaming

audio file across the Internet. Moreover, once a certain asset has been downloaded to a client machine, the CPU must be powerful enough to handle the sophistication of the asset. Finally, information about an end user's machine is unknown to the developer before the application launches. Because of these three reasons, developing just the client side application of a client-server application is a slow and intricate process for the developers.

Examples of Possible Client Machine Platform Variations:

Figures 5A– 5C are tables depicting the client platform variability by organizing into three tables some of the possible characteristics for client platforms. These tables shows more explicitly than does Figure 4 in underscoring the platform proliferation problem face by a developer in coding a client side application.

In particular, Figure 5A uses two aspects of platform characteristics: the extrinsic factor of connection bandwidth available to the client machine 501, and the intrinsic factor of memory size of the client machine 502. On the one hand, the bandwidth characteristics is differentiated into High 503 and Low 504. On the other hand, the memory size characteristics is differentiated into Large 505 and Small 506. Altogether, this table 522 presents four possibilities of client platform characteristics (setups): High/Large 511, High/Small 513, Low/Large 512 and Low/Small 514.

In Figure 5 B the connection bandwidth 501 is also used as an aspect of platform characteristics as Figure 5 A. But in Figure 5 B, the aspect of CPU speed 509 is used. The aspect of CPU speed is differentiated into Fast 507 and Slow 508. Altogether, this table 533 presents four possibilities of client platform characteristics (setups): High/Fast 581, High/Slow 583, Low/Fast 582, and Low/Slow 584.

In Figure 5 C, a table lists several more aspects of platform characteristics, including the aspects of connection bandwidth 501, memory size 502, and CPU speed 509. The other aspects are persistent storage types 571 and software functions available on the client machine 572.

In Figure 6, a preferred embodiment of the present invention is illustrated in detail. To take advantage of the programming model offered by the present invention, a developer decides on the types of assets 651~656 that will be used in the client side application of the entire interactive on-line application under design. Also, the developer can offer a gradation of quality layers for each asset. For example, for each image file asset, the developer can easily offer at least two layers of quality: a black and white image file asset as a quality layer, and a full color image file asset as the other quality layer.

In Figure 6, an entire instance of an interactive on-line application (680 and 684 combined) comprises the server side application 680 running on the on-line application server 300, and the client side application 684 running on top of the browser 274 for the client 204. Importantly, the present invention simplifies the development process of the client side application.

In addition to the client side application itself, the developer also supplies the Manifest 697 that records the list of assets and their specific properties. The assets 651~656 used by the client application 684 might be located in one location 200 as shown, or they could be scattered throughout the Internet. Fortunately, because these assets' locations are recorded in the Manifest 697, the application can locate and use these assets by referring to the Manifest 697.

In Figure 6, three clients 204, 206 and 208 are shown across the Internet 202 from an on-line application server 300 and an HTTP server 200. These three clients are drawn with different shapes to indicate three different machine platforms. Firstly, these clients differ in the types of persistent local storages. Client 204 has both a hard disk 664 and a CD-ROM 665, client 206 has only a hard disk 667, and client 208 has no persistent local storage at all. Secondly, Client 206 has a more DRAM than Client 208 does. Finally, Client 206 has low connection bandwidth and Client 208 has high connection bandwidth.

With reference to Figure 6, each client is running its own Asset Manager tailored to the specific strength and weakness of its machine platform. Client 204 is running the Asset Manager 634 that takes advantage of the availability of local storages 664 and 665.

Client 206 is running the Asset Manager 636 that takes advantage of the large DRAM size to minimize repeated download through Client (206)'s low bandwidth connection 696. The Asset Manager 636 downloads the very minimum number of assets having the lowest quality that still allow the application to run in a visually rational manner. And as the application runs, the Asset Manager 636 downloads other assets in the background so that the end user can eventually experience high quality for each asset. Client 208 is running the Asset Manager 638 that takes advantage of the high connection bandwidth 698 and minimizes using the DRAM (the browser cache 648 in this case). This Asset Manager 638 aggressively purges non-immediate assets from the browser cache to continually create room for the needed assets that can be speedily downloaded through the high bandwidth connection 698.

Figure 7 illustrates the concept of multiple storage images of an asset. An asset 777 used by an interactive on-line application (680 and 684) is first stored in certain HTTP server 200. As this asset 777 is downloaded to a client 204 having multiple storages such

as a hard disk 664 and a CD-ROM 665, the asset 777 is stored and retrieved differently depending on the storage type. The asset 777 can be represented by a storage image 755 in the hard disk 665, by a reference to a read-only storage image 744 in the CD-ROM 665, or by a storage image 766 in the browser cache 644. The client's Asset Manager 634 therefore keeps records of how and where an original asset is stored by treating the asset 777 as having different storage images in different storages.

The Programming Model provided by the Present Invention:

Figure 8 A depicts the procedures of making a Manifest 697 by a developer who follows the present invention's programming model. In essence, Figure 8 A provides the schematics of the present invention's programming model for making a Manifest 697. First, the developer specifies all of the assets needed by the application 384. Then for each asset, the developer designs several levels of quality 805. Also, each layer of quality for any type of asset is given a quality tag. For example, if asset A is an image file, then the developer naturally can have at least two layers of quality, i.e., monochrome and full color. After each asset is further differentiated into levels of quality 805, the developer then determine the minimum sets of assets required to run the application at a particular point of running the application. For example, the developer uses priority tags to classify all assets into assets designated with priority 1 (for highest priority), with priority 2 (for the next highest priority), and so on. As a result, in step 808 the developer codes into asset management policies for the logistics of how the application utilizes the assets. Referring again to Figure 6, these policies can be coded in the server side application 680, or in the client side application 684, or spread between both (680 and 684).

In conjunction with coding policies expressed via priority levels, the developer also records the information about asset quality 804 and asset priority level 806 in the Manifest

697. In the final step 877, the Asset Manager 634 refers to the Manifest 697 of the application to interact with the client side application 684.

It should be apparent to one of ordinary skill in the art that the actions of the developer described supra could be performed manually, or automatedly using computers, or partly manually and partly automatedly. The method produces similar results irrespective of method implementation details.

Figure 8 B shows in more detail the information types and formats contained within a Manifest 697. For each asset listed in the Manifest 697, a unique ID number 851 is given. The other parameters of each asset comprise asset data type 852, unique symbolic name 853 for coding the application, asset file size 855, check sum 856, quality layer 857, URL (Uniform Resource Locator) 858 indicating the storage location of the asset, and priority level 859.

The Internal Components of the Asset Manager:

The Asset Manager is a script interpreting engine that supports and exports an interface containing functions for managing the assets needed by the client side application. The client side application invokes the functions of the Asset Manager to manage all of the assets needed to run the application. As the application launches, the Asset Manager object and the Asset Registry object are instantiated. Also the Asset Manager comprises a group of data type factories objects, with one data object type factory per each assets types supported by the Asset Manager. These data type factories are also loaded together with the Asset Manager. For example, some of the common data type factories are the text data type factory for managing text assets, the image data type factory for managing image assets, the

audio data type factory for managing audio assets, and so on. All of these factory types are registered in the Asset registry.

When the application requests a particular asset, e.g., an asset with ID equals to 3, the Asset Manager looks in the Manifest to find that the data type of asset having ID which is equals to 3 is an image data type. Then the Asset Manager calls upon the image data type factory to create an object in response to the request. This asset is stored in the central location (the Asset Store) for storing all objects created by data type factories in response to request.

Specifically, with reference to Figure 9, the internal components of the Asset Manager 634 are discussed. First the client side application of a client-server application, say an interactive on-line application, is coded according to the present invention's programming model. As the client side application (684 of Figure 6) launched, both the Asset Manager object 634 and the asset registry object 920 are instantiated, while the Manifest 697 becomes downloaded from the application server 300. Instantiation of computer objects is well known in the computing arts. The Asset Manager object 634 comprises a set of data object factories 901~905, with typically one data object factory per asset type specified in the Manifest 697. These data object factories 901~905 are registered to the asset registry object 920. As the application runs, whenever an image asset is needed, the image data object factory 902 creates an image asset object 999 in a well known location called the Asset Store 940. The image player 942, the audio player 944, and the client side application check the Asset Store 940 for matching files and object types. For example, the image player 942 would find and play an image asset object 999 inside the Asset Store 940.

With reference to Figure 10, a flow chart is given for outlining the schematics of Figure 9. In the step 1002, as the application launches, the Manifest is loaded to the client. In step 1004, the application notifies the Asset Manager to load the asset with an ID of 3 for example. In step 1006, the Asset Manager searches in the Manifest to see that the asset having ID that is equal to 3 and finds the asset's data type to be the image data type. In the step 1008, the Asset Manager pulls from the asset registry the image data object factory in order to load this particular image data object. In the step 1010, the image data object factory creates an image object representing the image asset in memory. In the step 1012, the image data object factory adds the image object to the asset store.

Extensibility of Asset Types for the Asset Manager:

As discussed above with reference to Figure 9, the Asset Manager supports a group of data object factories, wherein each data object factory responsible for managing a specific type of asset. This particular design of the Asset Manager is a powerful feature of the present invention because the support for managing a new type of assets can be easily added to the Asset Manager without affecting existing codes. As support for any new type of assets is added to the Asset Manager, no change to existing code of the Asset Manager is necessary. To add support to any new type of assets, the mere addition of a new data type factory to the Asset Manager would suffice. Consequently, this design of the Asset Manager enables the extensibility of asset types.

For example, with reference to Figure 9, if a new type of asset that needed to be supported by the Asset Manager is a "movie" asset, the developer simply codes the movie data object factory 905 for inclusion in the group of data object factories for the Asset Manager 634. Then the developer makes available a movie player 946 on the client machine

for playing the movie assets. New code for a new type of asset can be added without affecting any existing codes of the client side application or the Asset Manager.

Another important aspect of the present invention's data type extensibility is the ability to support assets of very large sizes such as a streaming data object, without affecting existing code. For example, if a very long download of a very large size audio asset becomes necessary in running the client side application, the Asset Manager can create an illusion that a download has been completed. For a streaming asset such as an audio asset, the data object representing this audio asset does not have to be fully instantiated in the Asset Store by the audio data type factory. That is to say, partially instantiated asset data object can be accessed by the client side application as if the asset data object were completely instantiated. An audio player can actually begin to play the partially instantiated audio data object while the audio data type factory of the Asset Manager continues to instantiate the rest of the audio data object.

The Asset Store and switching Applications:

The Asset Store plays a facilitating role when an end user is switching between different interactive on-line applications. Specifically, the Asset Store can store all of the instantiated assets shared by various client side applications. Then, any of these client side applications can directly access and use these instantiated assets.

With reference to Figure 11, a flow chart is presented to show how the Asset Store can be applied for playing the game of JEOPARDY (TM) on-line. For example, if a user wants to play the game of JEOPARDY (TM) on-line with some other players, the user first logs on-line to launch the lobby application of JEOPARDY (TM) to meet potential opponents, as outlined in steps 1102, 1104 and 1106. Lobby applications are well known

in the online computer gaming arts. Then, in the step 1108, once the user has decided who the other players are, the game application for Jeopardy is launched in the step 1110. Finally, as a game is finished in the step 1114, the user can go back to the "lobby" to find some other new opponents by launching the lobby application once again, as outlined in sequential steps of 1118, 1122, then 1104. Throughout the whole process, the user would have switched between the lobby application and the game application at least twice, as in steps 1124 and 1122. The Asset Store facilitates the switching between these two applications because it retains all of the assets needed by the lobby application when the user switches to the game application. So when the user switches back to the lobby application from the game application, the assets needed by the lobby application are already in place in the Asset Store, ready to be used. Similarly, because the Asset Store retains all of the assets needed by the game application, as the user switches from the lobby application back to the game application, the game application can start right away without waiting for its assets to get downloaded. As a result, the Asset Store enhances the user's on-line experience of playing JEOPARDY (TM) because the Asset Store facilitates the switching between the lobby application and the game application.

The foregoing descriptions of specific embodiments of the present invention have been presented for purposes of illustration and description. They are not intended to be exhaustive or to limit the invention to the precise forms disclosed, and obviously many modifications and variations are possible in light of the above teaching. The embodiments were chosen and described in order to best explain the principles of the invention and its practical application, to thereby enable others skilled in the art to best utilize the invention and various embodiments with various modification as are suited to the particular use contemplated. The scope of the invention is intended to be defined by the Claims appended hereto and their equivalents.

CLAIMS

What is claimed is:

1. For a client-server system having a plurality of different client platforms, a computer implemented method of generating a client side application of a client-server application, said method comprising:

providing a programming model to be followed by developers for developing said client side application;

providing a programming model to be followed by developers for upgrading said client side application;

generating a file associated with said client side application to record an asset required by said client side application, and wherein at least one of said assets is a data object; and

managing programming logistics for delivering assets required by said client side application, and wherein said managing step is executed on at least one client machine of said client-server system by a platform-specific system level software component of a client machine.

2. The computer implemented method of Claim 1, wherein said programming model requires steps comprising:

specifying multiple layers of quality for at least one of said assets, wherein said asset is a data object having data types selectable from a group comprising video data objects, audio data objects, code data objects, data chunk data objects, graphic data objects, image data objects;

setting a level of priority for delivering at least one of said assets to said client machine;

centralizing in said file information associated with at least one of said assets required by said client application;

designing asset delivering policies with said centralized information of said file; and

providing extensibility of new asset data types, wherein adding new asset data types does not require changing existing code.

3. The computer implemented method of Claim 2, wherein said multiple layers of quality comprise:

a plurality of audio quality layers ranging from low fidelity to high fidelity; and

a plurality of image quality layers ranging from monochrome to full color.

4. The computer implemented method of Claim 2, wherein said priority levels are ranked by positive integers, and an integer one designates an asset having the highest download priority.

5. The computer implemented method of Claim 2, wherein for each said asset, said centralized information in said file comprises:

a field of unique ID number;

a field of data type; and

a field of unique symbolic name.

6. The computer implemented method of Claim 2, wherein said policies need to be coded only once for each client platform, and wherein said policies are coded with said download priorities specified in said file.

7. The computer implemented method of Claim 2, wherein said addition of new asset data types is implemented by adding a data object factory per one new asset data type in

said software component without affecting existing code of said software component and said client side application.

8. The computer implemented method of Claim 1, wherein said managing step further comprises the steps of:

customizing said asset management software component for each type of client platform, wherein said software component manages asset delivery for said client application running on each client machine of said client-server system;

fetching said text file to each said client machine;

prioritizing download ordering of assets with reference to said file;

unifying multiple storage images of each said asset, wherein each of said storage images represents said asset in a distinct persistent memory storage location;

storing in a central memory location all assets shared by a plurality of client applications, wherein said central memory location is termed an Asset Store;

minimizing asset download to avoid interrupting sessions of said client side application session;

9. The computer implemented method of Claim 8, wherein each said type of client platform having characteristics ranges:

from high to low connection bandwidths;

from large to small dynamic memory sizes;

from high to low CPU processing speeds; and

from several to no persistent memory storages.

10. The computer implemented method of Claim 8, wherein said client application can start before all of said required assets are available, and wherein said software component continues background downloading not-yet downloaded assets.

11. The computer implemented method of Claim 8, wherein said multiple storage images are uniquely keyed and encoded in addition to said asset located on a HTTP Web server, and wherein said software component use available persistently stored storage images of said assets to avoid downloading said asset from said HTTP Web server.

12. The computer implemented method of Claim 8, wherein said shared assets minimizes latency in switching between applications.

13. The computer implemented method of Claim 8, wherein said minimizing step comprises:

searching said text file for all assets having priority levels lower than assets having the highest priority level;

downloading said assets having a highest priority level;

starting said client application as soon as said assets having the highest priority level are completely downloaded;

downloading said lower priority assets in the background while said client application is in session; and

downloading said lower priority assets while said client application is in between sessions.

14. For a client-server system having a plurality of client machines from different client platforms, a computer-readable medium having stored thereon instructions for causing said client machines to deliver assets required by a client side application of a client-server application, comprising code for supporting:

a plurality of assets fetched from a plurality of servers;

a server component generating a file to list said required assets and to specify properties associated with said assets;

a management component for delivering said assets in accordance to said plurality of policies; and

an extensible set of data object factories, wherein each of said data object factories is associated with a distinct asset data type.

15. The computer-readable medium of Claim 14, wherein each said asset is a data object selectable from a group comprising video data objects, audio data objects, code data objects, data chunk data objects, graphic data objects, and image data objects.

16. The computer-readable medium of Claim 14, wherein said file is a manifest associated with said client side application, said manifest comprising:

a field of unique ID number for each said asset;

a field of data type for each said asset; and

a field of unique symbolic name for each said asset.

17. The computer-readable medium of Claim 14, wherein the priority levels assignable to each of said assets are designated by positive integers, wherein an integer one designates an asset with the highest download priority level.

18. The computer-readable medium of Claim 14, wherein for at least one of said client machines, said policies specify delivering a necessary platform-specific subset of said assets at a necessary point in time in order for said client side application to proceed in a visually appealing and rational manner.

19. The computer-readable medium of Claim 14, wherein said management component downloads said assets to at least one of said client machines according to said policies, and wherein said management component fetches said assets directly from a plurality of local persistent memory locations, provided said assets have been downloaded and stored in said plurality of local persistent memory locations.

20. The computer-readable medium of Claim 14, wherein at least one of said data object factories, in the dynamic memory of said client machines, instantiates a data object representing an asset with the same data type as said data object.

21. The computer-readable medium of Claim 20, wherein at least one of said instantiated data object is stored in a common dynamic memory location, wherein another client side application requiring such said data object can directly access said data object without any further data object instantiation.

22. The computer-readable medium of Claim 14, wherein said servers are HTTP Web servers.

23. The computer-readable medium of Claim 14, further comprising a plurality of priority levels assigned to said assets.

24. The computer-readable medium of Claim 14, further comprising a plurality of policies for delivering said assets to said client machines.

25. The computer implemented method of Claim 2, wherein for each said asset, said centralized information in said file further comprises:

a field of URL;

a field of quality;

a field of checksum; and

a field of download priority level.

26. The computer-readable medium of Claim 14, wherein said file is a manifest associated with said client side application, said manifest further comprising:

a field of URL for each said asset;

a field of asset file size for each said asset;

a field of quality for each said asset;

a field of checksum for each said asset; and

a field of download priority level for each said asset.

Figure 1
(prior art)

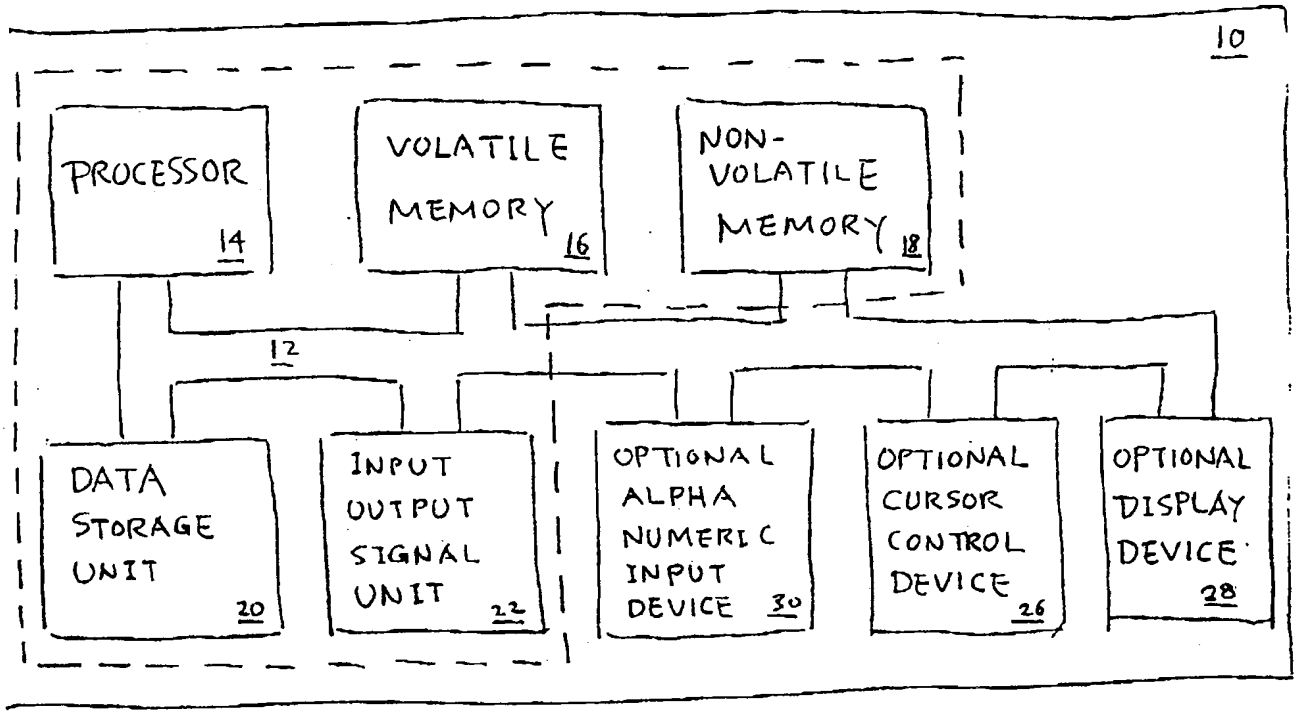


Figure 2
(prior art)

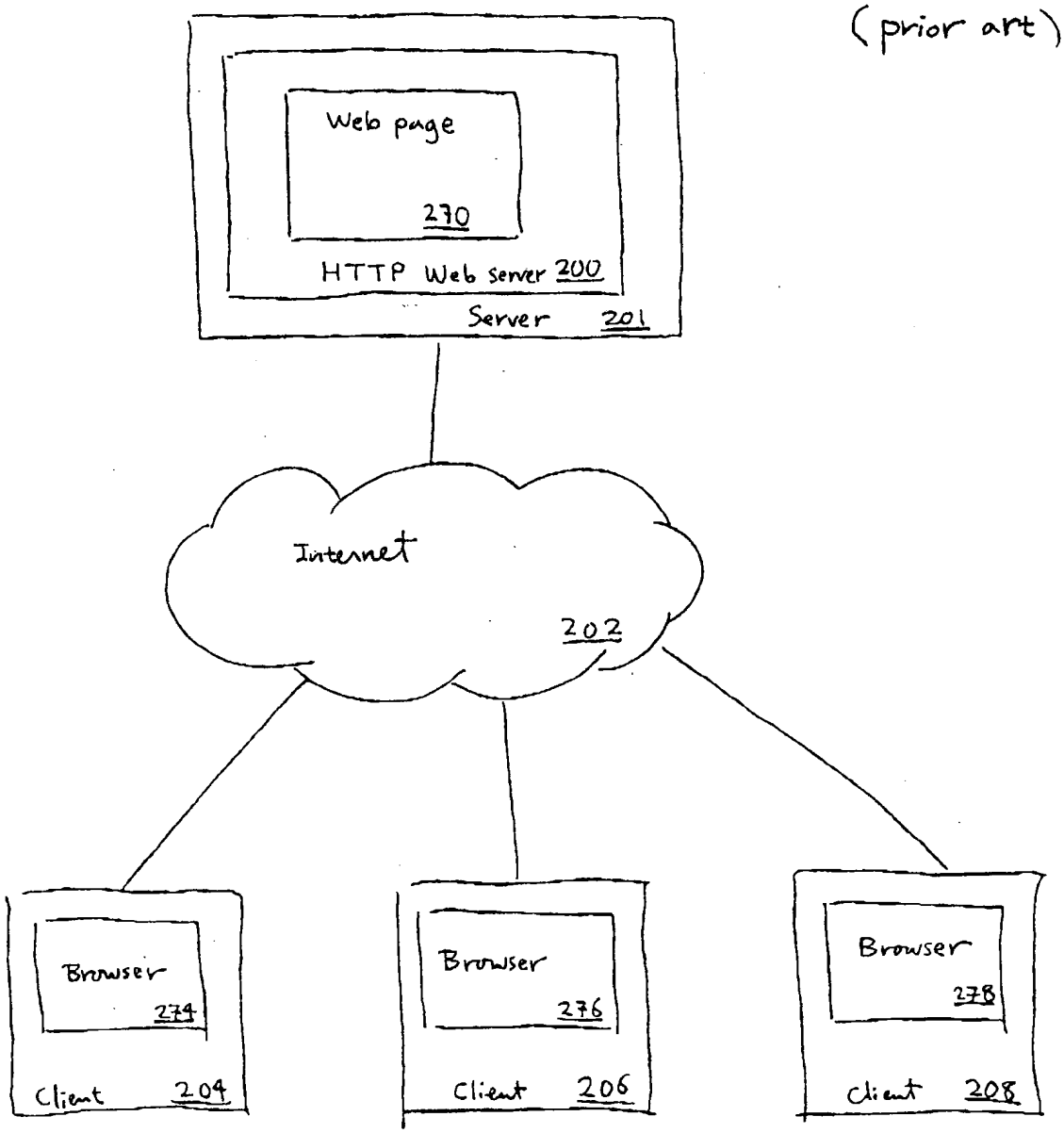


Figure 3
(prior art)

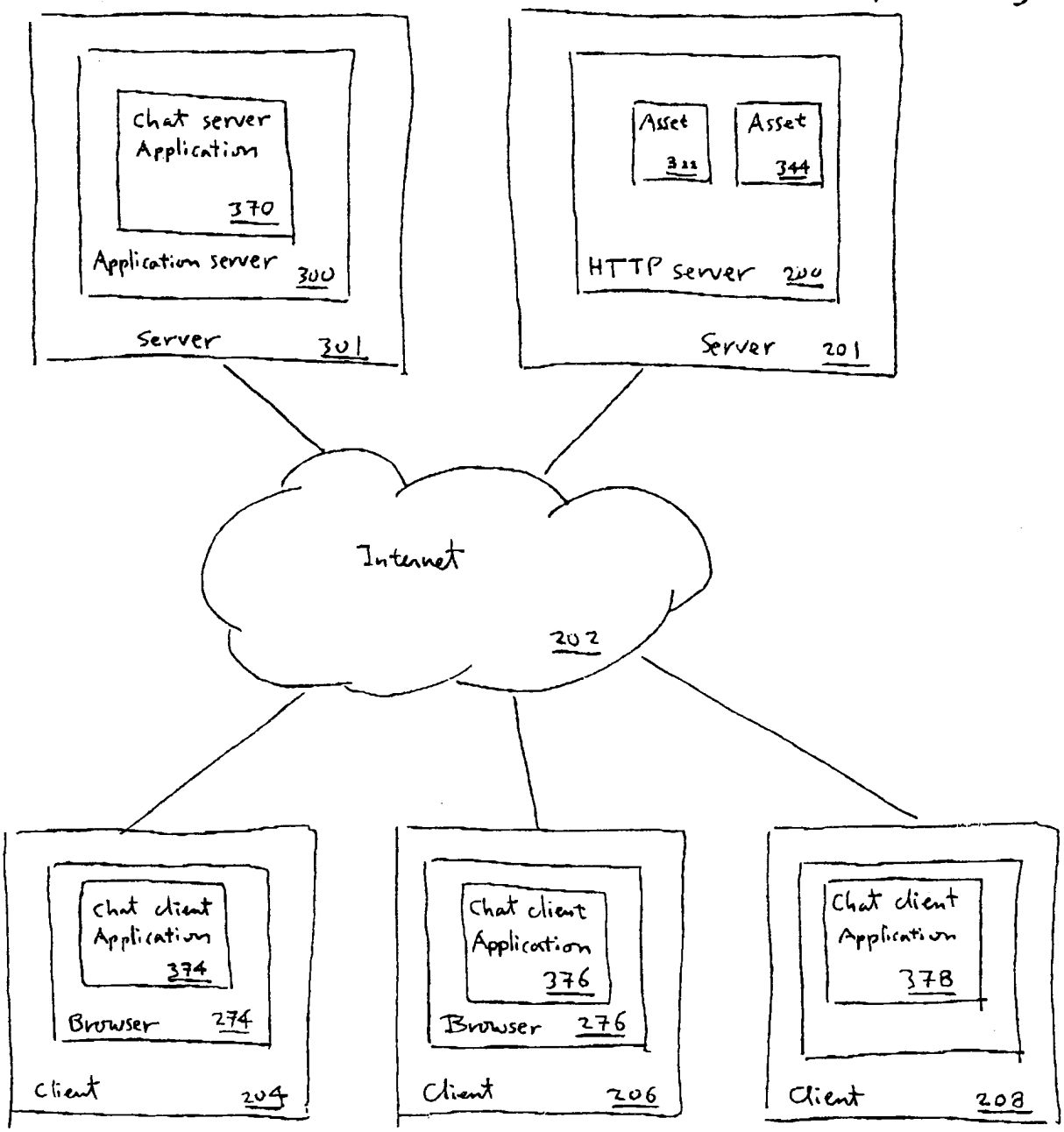


Figure 4

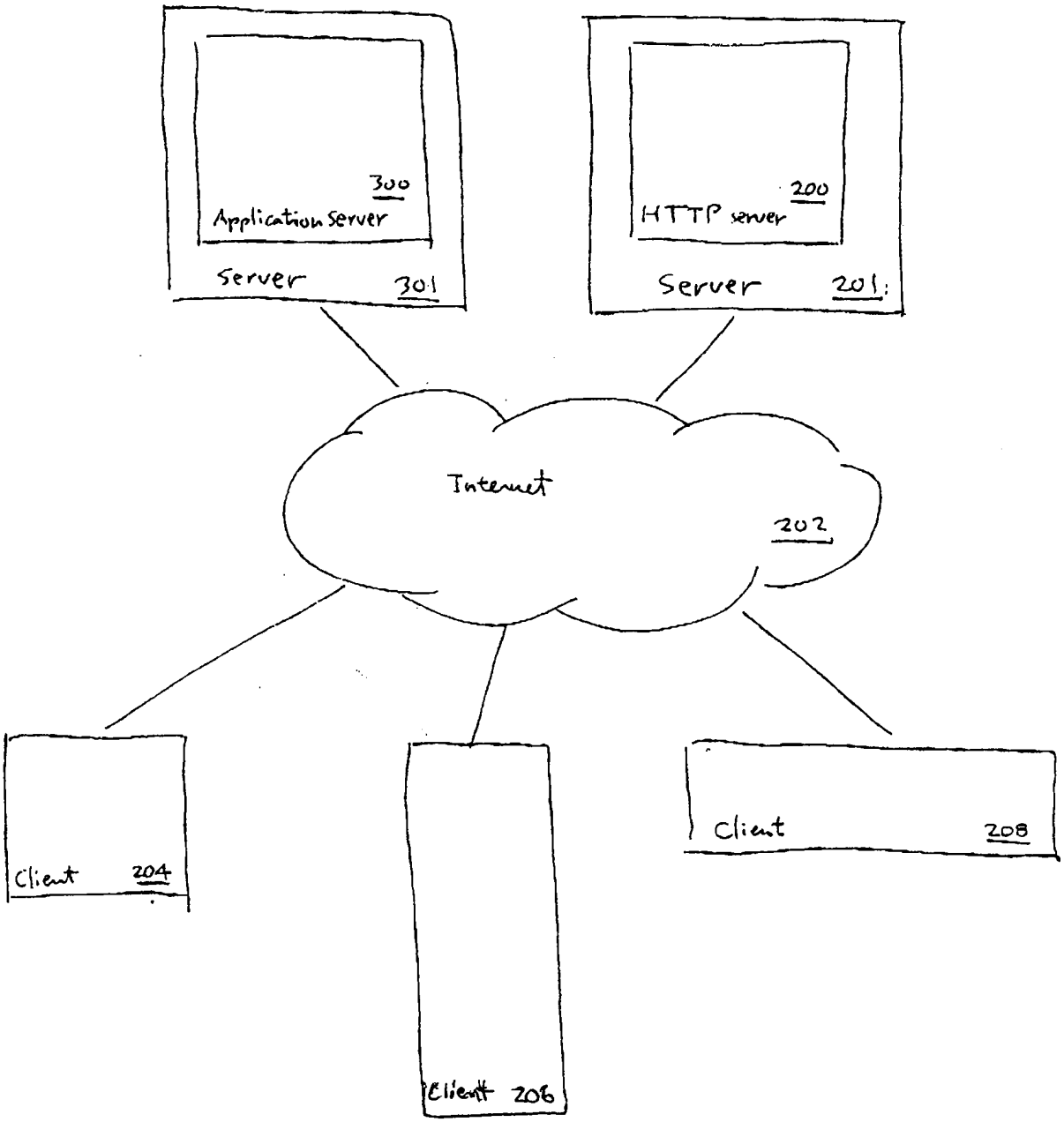
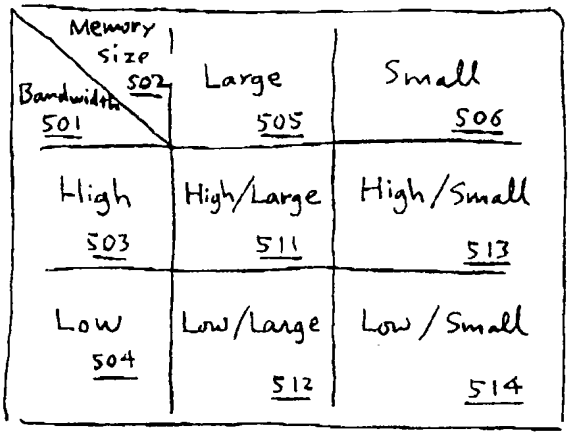
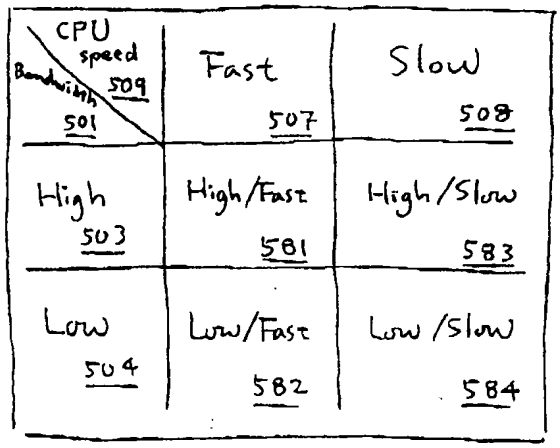


Figure 5A



Bandwidth 501 \ Memory size 502	Large 505	Small 506
High 503	High/Large 511	High/Small 513
Low 504	Low/Large 512	Low/Small 514

Figure 5B



Bandwidth 501 \ CPU speed 509	Fast 507	Slow 508
High 503	High/Fast 581	High/Slow 583
Low 504	Low/Fast 582	Low/Slow 584

Figure 5C

Client Machine Characteristics	Range of Variation
Connection Bandwidth 501	High ⟷ Low
Memory Size 502	Large ⟷ Small
CPU Speed 509	Fast ⟷ Slow
Persistent Storage 523	{HD, CD-ROM} ⟷ none
Available 3rd Party Application	{audio player, video player} ⟷ none

Figure 6

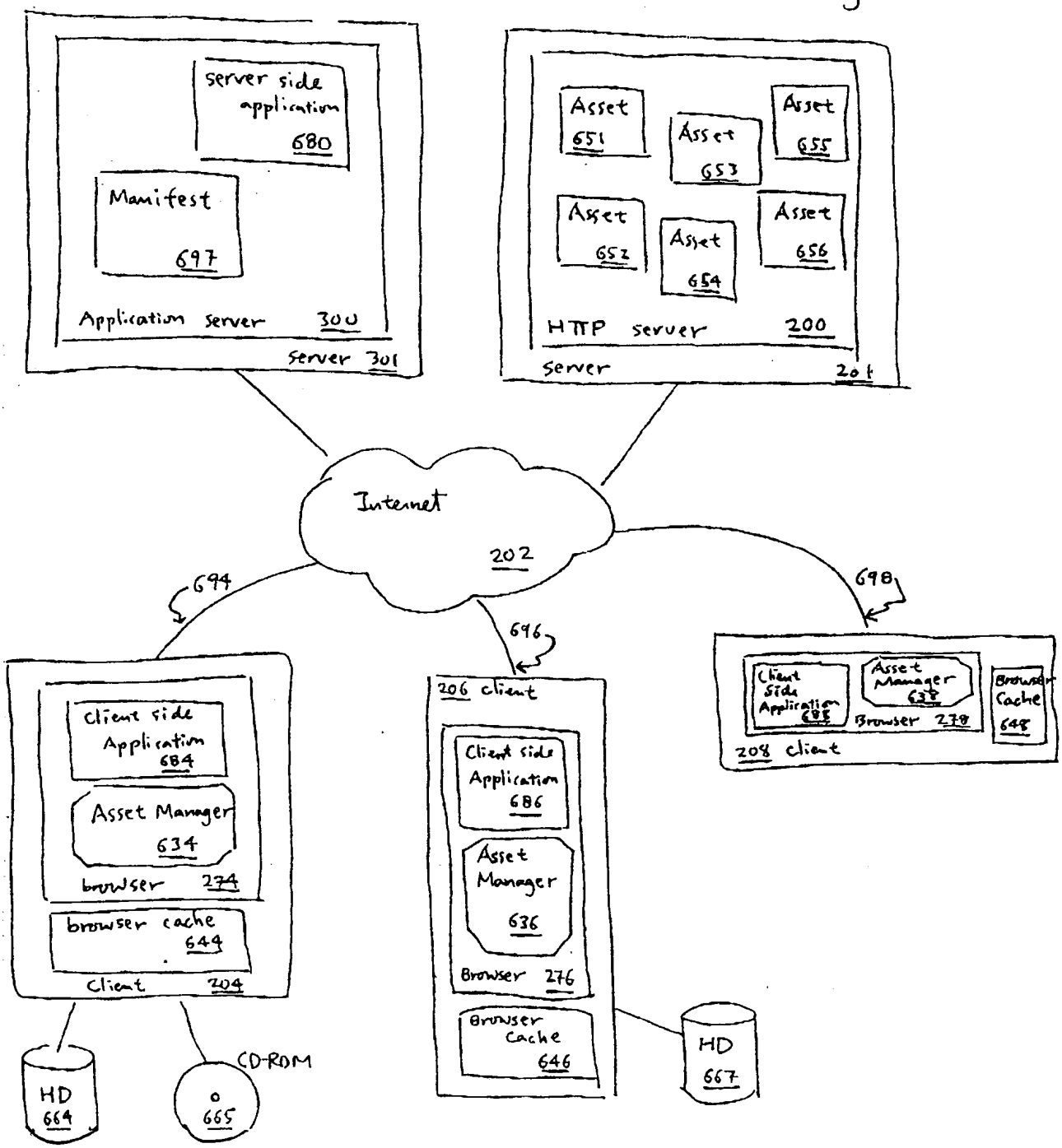


Figure 7

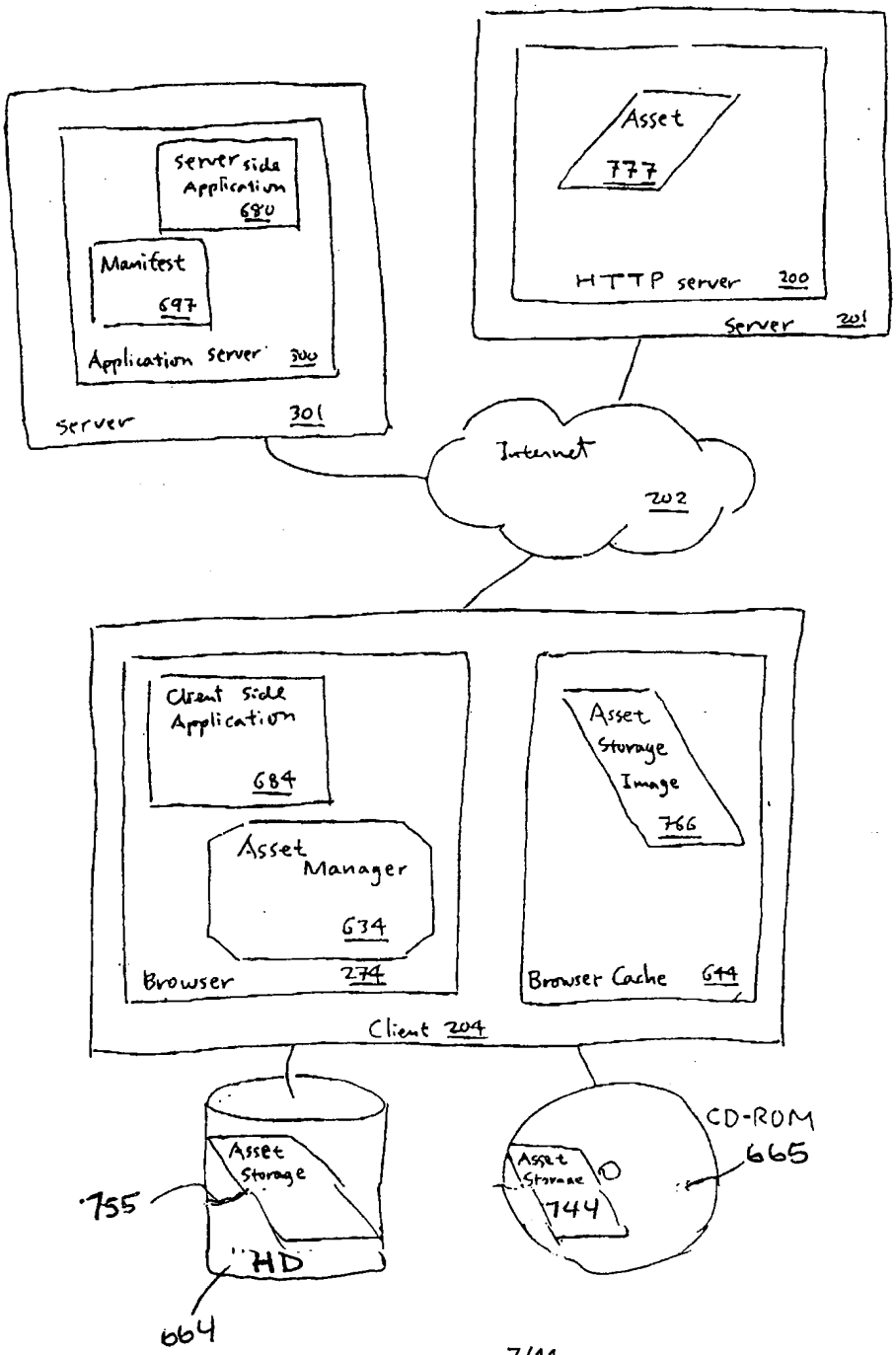


Figure 8A

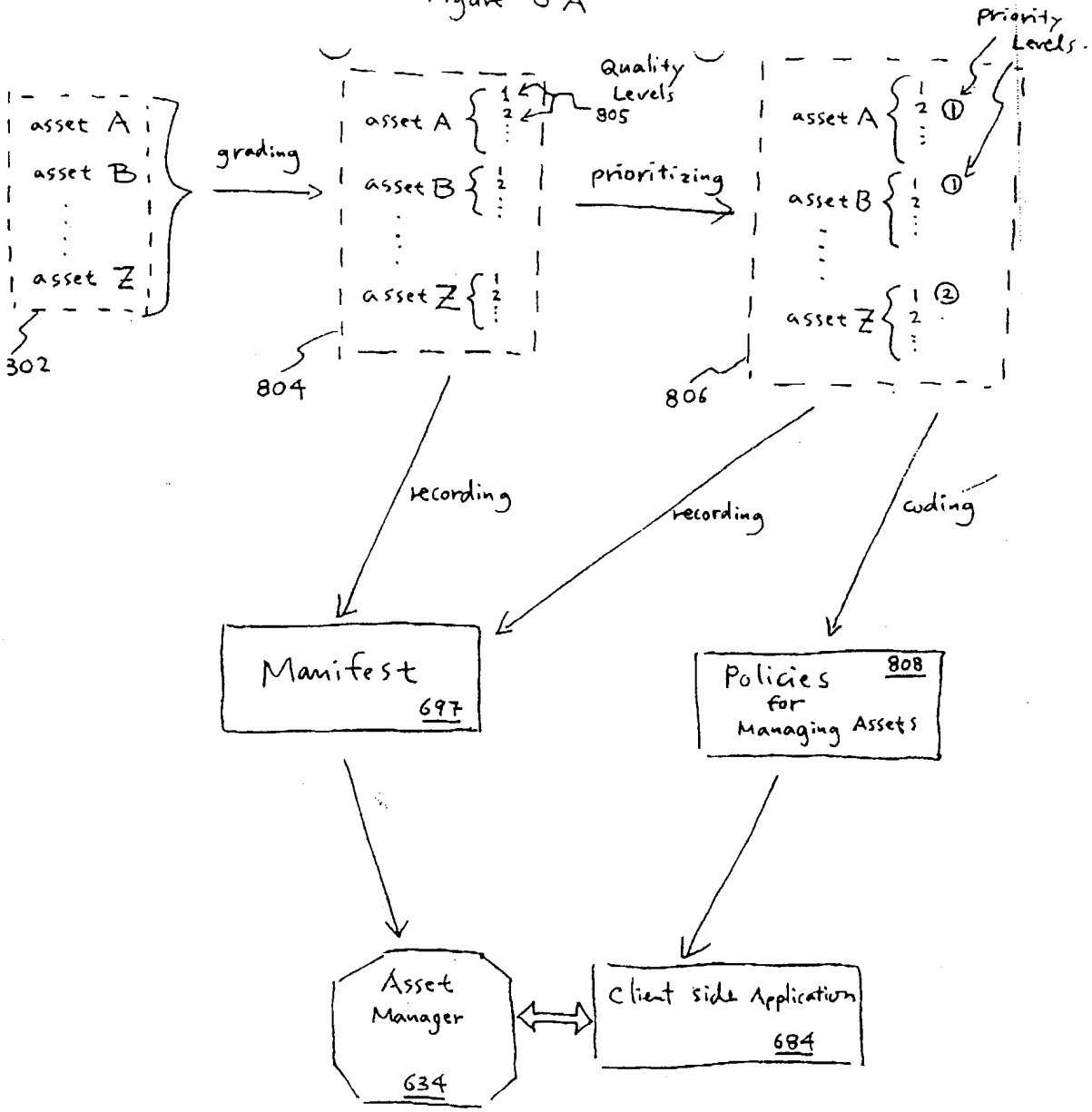


Figure 8B

	ID number	Data Type	Symbolic Name	URL	Quality	Checksum	Priority Level	File Size
Asset Ω	851	852	853	858	857	856	859	855

Figure 9

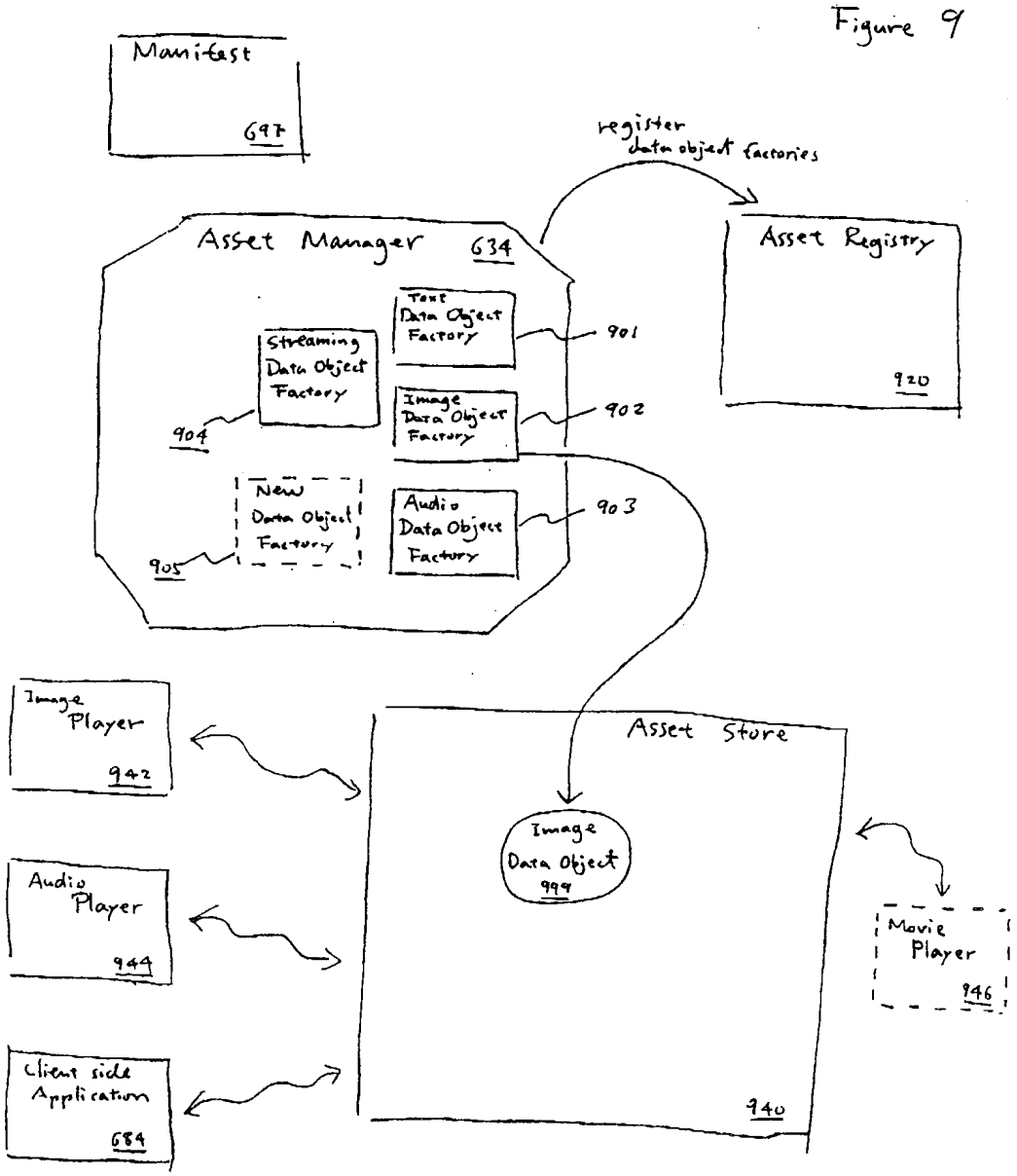


Figure 10

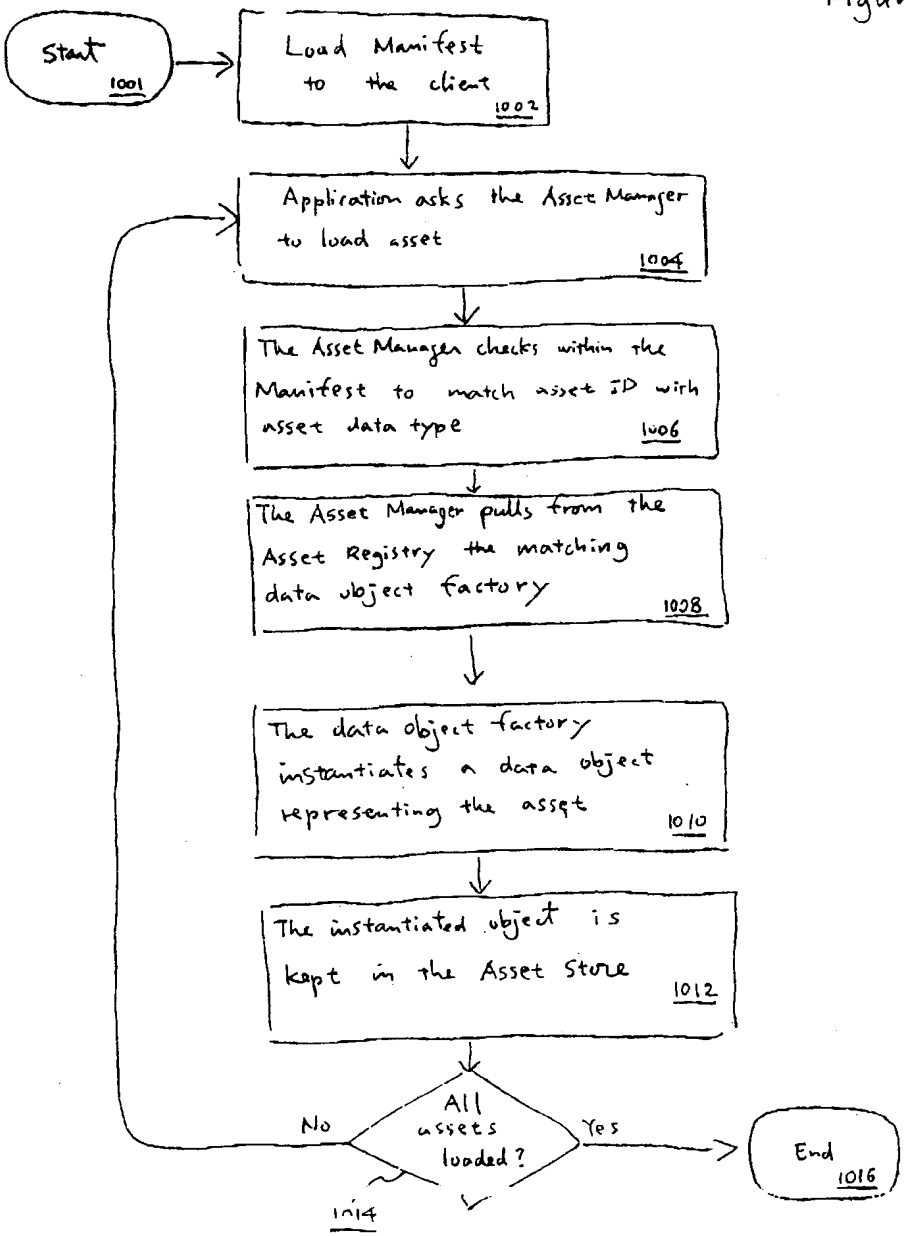
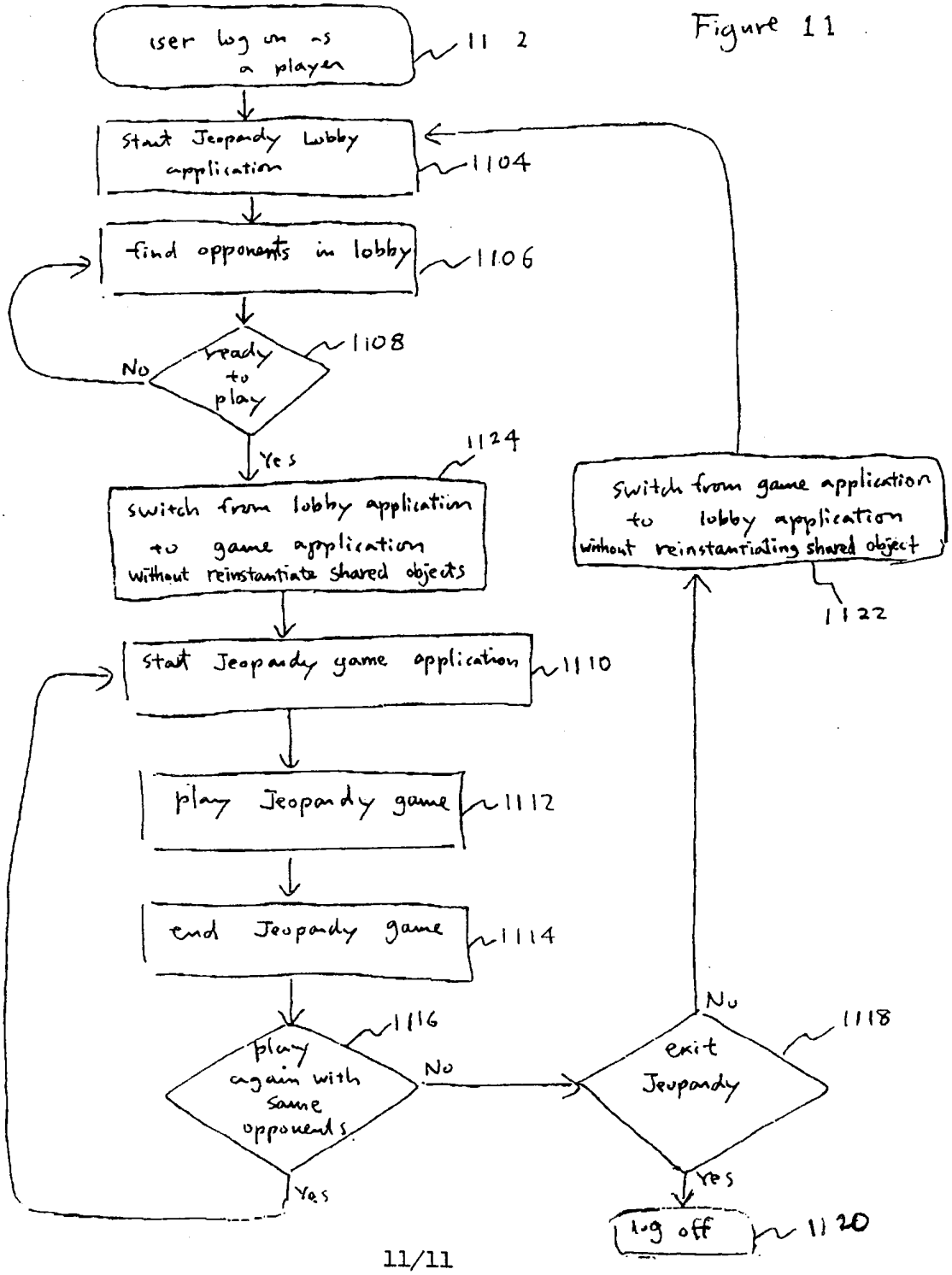


Figure 11



INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 00/03346

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F9/44

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, INSPEC, IBM-TDB

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5 815 711 A (SAKAMOTO KAZUHIKO ET AL) 29 September 1998 (1998-09-29) abstract column 1, line 1 -column 2, line 11 column 2, line 55 -column 4, line 67 claim 1 ---	1,14
A	NOBORU AKIMA ET AL: "INDUSTRIALIZING SOFTWARE DEVELOPMENT: A JAPANESE APPROACH" IEEE SOFTWARE,US,IEEE COMPUTER SOCIETY. LOS ALAMITOS, vol. 6, no. 2, 1 March 1989 (1989-03-01), pages 13-21, XP000007936 ISSN: 0740-7459 the whole document --- -/--	1,14

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
11 July 2000	18/07/2000

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Beltrán-Escavy, J

1

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 00/03346

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
A	US 5 655 091 A (DRAKO DEAN M ET AL) 5 August 1997 (1997-08-05) abstract column 1, line 1 -column 2, line 55 column 3, line 40 -column 4, line 30 column 6, line 3 -column 6, line 32 claim 1	1-3,8, 11,14, 15,22
A	EP 0 649 121 A (IBM) 19 April 1995 (1995-04-19) abstract page 3, line 1 -page 6, line 43 page 7, line 54 -page 8, line 25 claims 6,7	1,8,11, 14,15,22
A	WO 97 15018 A (BELL COMMUNICATIONS RES) 24 April 1997 (1997-04-24) page 1, line 1 -page 5, line 3 claim 1	1,8,11, 14,15,22

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/US 00/03346

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5815711 A	29-09-1998	JP 8137678 A	31-05-1996
US 5655091 A	05-08-1997	US 5446866 A	29-08-1995
		DE 69216379 D	13-02-1997
		EP 0553549 A	04-08-1993
		JP 6059989 A	04-03-1994
EP 0649121 A	19-04-1995	AT 189074 T	15-02-2000
		DE 69422679 D	24-02-2000
		DE 69422679 T	06-07-2000
		JP 7175868 A	14-07-1995
		JP 10207965 A	07-08-1998
		KR 143358 B	17-08-1998
		US 5734719 A	31-03-1998
WO 9715018 A	24-04-1997	NONE	

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

CORRECTED VERSION

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
17 August 2000 (17.08.2000)

PCT

(10) International Publication Number
WO 00/48073 A1

(51) International Patent Classification[7]: G06F 9/44

(21) International Application Number: PCT/US00/03346

(22) International Filing Date: 9 February 2000 (09.02.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/248,180 9 February 1999 (09.02.1999) US

(63) Related by continuation (CON) or continuation-in-part (CIP) to earlier application:
US 09/248,180 (CON)
Filed on 9 February 1999 (09.02.1999)

(71) Applicant *(for all designated States except US)*: **HEARME** [US/US]; 665 Clyde Avenue, Mountain View, CA 94043 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: **ROSKOWSKI, Steven, G.** [US/US]; 1680 English Court, San Jose, CA 95129 (US). **NA, Piaw** [US/US]; 13376 Ronnie Way, Saratoga, CA 95070 (US).

(74) Agents: **GALLENSON, Mavis, S.** et al.; Ladas & Parry, Suite 2100, 5670 Wilshire Boulevard, Los Angeles, CA 90036-5679 (US).

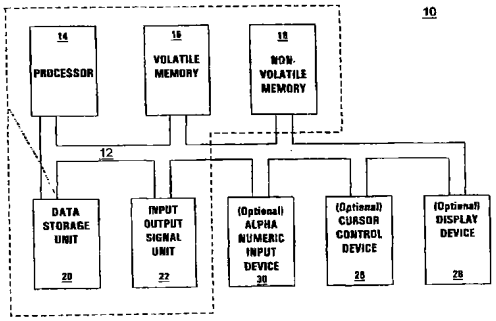
(81) Designated States *(national)*: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

[Continued on next page]

(54) Title: METHOD AND APPARATUS FOR MANAGING ASSETS OF A CLIENT SIDE APPLICATION



(57) Abstract: The present invention simplifies the development of a client side application for a client-server application, wherein such a client side application typically runs on multiple clients having drastically different platform characteristics. To simplify the development process of a client side application, the present invention a) provides a programming model as a guide for code development, and b) supports a software component at each client machine's system level for asset management. The programming model guides the coding development of the client side application in a standardized and systematic manner. It also allows the developers to write codes that are platform-independent. Asset management is automated by a client system level software component (called the Asset Manager) running on each client's side. The Asset Manager of each client is tailored specifically for maximizing the capability of the client machine platform. Moreover, the Asset Manager of each client enables the development of client platform-independent code. In addition, the design of the Asset Manager via data type factories allows extensibility of media asset types; a new asset data type can easily be supported by simply adding a new data type factory to the Asset Manager without rewriting existing code. Finally, the Asset Manager uses a central location called the Asset Store to store assets shared by different client side applications, thus facilitating the speed of switching among these applications.

WO 00/48073 A1

(48) Date of publication of this corrected version:
4 October 2001

(15) Information about Correction:
see PCT Gazette No. 40/2001 of 4 October 2001, Section II

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

METHOD AND APPARATUS FOR MANAGING ASSETS OF A CLIENT SIDE APPLICATION

FIELD OF THE INVENTION

The present invention relates to the client side development process of a client-server application. Specifically, the present invention offers a platform-independent programming model for efficiently developing and upgrading such client side application. Moreover, for each distinct client platform, the present invention manages the platform-specific programming logistics for delivering to the client machine large size data objects (assets) required by the client side application

BACKGROUND OF THE INVENTION

Client-server applications prevalent on the Internet today are mostly information based: applications serving information to the browser for display. But at the same time, exciting possibilities exist for developing more sophisticated client-server applications in at least several directions. One possible direction for the future of client-server applications is interaction oriented. Instead of browsing rather passively for information, the end users might want to be able interact with other users on-line (e.g. on-line gaming, on-line chatting). Another possible direction for the future of client-server applications is presentation oriented. The presentation of information would no longer be confined to the usual simple text and graphics data types. The focus of the application development would be on taking multi-media to a new level. Novel and effective formats of data types would continually be created to engage the users' interest and enhance the users' experience with richly presented information. Naturally, applications can also be both: offering high level of interactivity and presenting media rich information.

Commercially, developing more sophisticated client-server applications holds exciting promises. The user sessions for interactive applications are usually much longer than a browsing session. So interactive applications might offer an intriguing and effective medium for advertising. On the other hand, Web sites having rich media contents can attract more user visits. So richly presented applications also offers a potentially attractive medium for advertising.

However, developing these applications has been for the most part a slow process because of client platform proliferation. This is especially true for the client side application development process of a client-server application. So, before developers build interactive on-line applications, they must face and overcome a variety of problems in developing the client side application. What stands out among these problems is the proliferation of client machine platforms having drastically different characteristics such as CPU performance, connection bandwidth, persistent storage types.

Notwithstanding the existing variety of client platforms available now, the trend of new and innovative client platforms will surely continue unabated. Witness recently the introduction of a variety of set top boxes, network PC's, and handheld electronic devices. Thus, a urgent need arises for containing and handling the complexity of accommodating different client platforms in developing the client side applications. This is the hardware problem for developing client side applications.

Specifically, the proliferation of client platforms is a problem for developers of client side applications because no developer can control what an end user would use as a client machine. The developers are thus ignorant about the client platform characteristics on the client side. In writing any client side application, this ignorance would have to be compensated by extensive preparation on developers' part to ensure that the client side

application will work with any type (or at least most types) of client platforms in existence. Furthermore, given that each client side application can be designed for very different objectives, codes accommodating different client platforms in one client side application might not be viable for another type of client side application. A new set of codes might be necessary to accommodate different client platforms in a new on-line application. Consequently, developing client side applications is a painstakingly slow process.

Besides the hardware issue above for developing client side applications, the developers face another problem, this time coming from the software side. This problem is the proliferation of new asset types (data object types) involved in developing these client side applications. The assets required by a client side application are traditionally media assets such as image data objects, audio data objects, and text data objects. But the term "asset" can more broadly refer to any large size data type object needed to be delivered to the client in a speedy and precise manner, just like the media assets.

This proliferation of asset types is inevitable because in order to enhance an end user's on-line experience, developers constantly strive to offer a great variety of presentation modes. An end user running an interactive on-line application might want to listen to a certain audio files, then watch some animation. Each of these data type objects requires specific handling. By allowing multiple ways for an end user to experience information, an application enhances the experience of that information.

Given that an end user's on-line experience is enhanced by using interesting mixture of data object types (asset types), introducing new data object types enhances the application. But as new data object types become available for incorporating into the application, code rewriting can become quite extensive and prohibitive to implement.

How are these new data objects displayed to the end users? And for a large size data object. how will it be downloaded? These are critical questions that needed to be answered in codes by the developers. Thus, what is further needed is a way to handle the proliferation of data type objects being created for client side applications.

To make matter worse for the developers, when considering these software issues juxtaposed with the hardware issue of whether client machines of a particular platform can display or process the new data objects, developing a client side application can easily become a daunting task. In particular, in developing a client side application, developers face the intricate problem resulting from running various data objects on various client platforms. Handling the permutations of different data objects and client platforms entails a labor intensive task for developing client side applications. This is the coordination problem of simultaneously coping with both the hardware and the software proliferation problems.

For the rationale given above, in developing and running an client side application, what is needed is a way to overcome the hardware problem of client platform proliferation , the software problem of data type object proliferation, and the intricate coordination problem of running different data type objects on different client platform. The present invention solves both the hardware problem and the software problem involved in developing and running a client side application. The present invention also solves the intricate problem of running various data type objects on various client platforms.

SUMMARY OF THE INVENTION

The present invention simplifies the development process of the client side application for client-server applications, where the client side application is run on client machines of drastically different platforms. The present invention provides for the developers a powerful and useful infrastructure to quickly and precisely develop a client side application. In the process, the present invention ensures delivery of media rich, and/or rapidly refreshed images, while freeing the developer to focus on content aspect of the application. Specifically, to simplify the development process of a client side application, the present invention a) provides a platform independent programming model as a guide for code development, and b) supports a extensible software component (called the Asset Manager) at each client machine's system level for asset management. The platform-independent programming model solves problems of proliferation client machine platforms. The extensible design of the Asset Manager also solves problems of proliferating asset data types. Together, the programming model and the Asset Manager solves the coordination problem of mixing various different hardware platforms with various different software data types.

Programming Model according to the Present Invention:

An important aspect of the present invention is its programming model. This programming model, when followed, ensures that any asset being used by the client side application is being downloaded in an efficient manner, and is being delivered in an aggressive manner. In developing an client side application, the developers need to lay out just once the logistics of how various assets will be used by the application. This programming model standardizes the language, the API (Application Programming Interface), and the procedures for expressing the logistics of managing and delivering

assets needed to run an client side application. Through this programming model, the application developers can also prioritize all assets according to their urgency to running program by priority levels. Thus, the present invention's programming model offers a standardized way to express policies and logistics of utilizing assets according to these assets' priority levels. For example, a asset may be assigned a high priority level, a medium priority level, or a low priority level. Then, the developers can code the policy in a standardized way by coding that high priority assets must be downloaded before the medium priority assets, and so on.

Asset Manager according to the Present Invention:

At the core of the present invention is a software component called the *Asset Manager* that is running on each client machine. The Asset Manager is a system level software component. It provides an extensible architecture for handling and downloading new assets and providing them for use by other components in the client system. Advantageously, the design of the Asset Manager via data type factories allows extensibility of asset data types: a new asset data type can easily be supported by simply adding a new data type factory to the Asset Manager without rewriting existing code. Moreover, the present invention offers one type of Asset Manager per client machine platform by tailoring one Asset Manager specifically for each distinct client machine platform in order to realize each client machine's full potential. In other words, an Asset Manager can be viewed as a platform-specific system level software component of its associated client machine.

In a sense, before the complexity of dealing with multiple client platforms ever reach the application developers, the Asset Manager on each client machine is designed to

stop that client machine's distinct platform characteristics from ever becoming an issue for the developers. Better still, the Asset Manager only has to be designed once.

For example, for a client machine having high connection bandwidth and small dynamic memory size, the Asset Manager of this client machine would be designed to capitalize on the high connection bandwidth while compensating for small dynamic memory size by minimizing memory access. The emphasis here is on repeated download and diligent memory purging. In the reverse case, for a client machine having low connection bandwidth and large memory size, the Asset Manager of this client machine would be designed to capitalize on the large memory size by caching a great deal of downloaded information and minimizing download. The emphasis here is on caching, minimizing downloading. Moreover, as the client application runs, the Asset Manager keeps recently accessed assets in the Asset Store, a location in dynamic memory. Then, other client applications requiring the same assets can directly access these assets in the Asset Store. As a result, the use of Asset Store by the Asset Manager minimizes redundant asset download, and reduces the latency in switching between different client applications.

An important role of the Asset Manager is coordinating and finding assets wherever they are, pulling them together, ensuring the necessary assets are present for the client side application to run. Managing the localities of assets is a subtle process. Assets may be inter alia located on local persistent disk memory, CD-ROM, or the server machine, or etc. The Asset Manager keeps track of various asset images in various locations by using a global information text file typically called the Manifest. The Manifest is a list of assets together with their corresponding asset properties such as description, priority levels, quality levels. By working in concert with the information specified on the Manifest (by the developers or automatically generated), the Asset Manager ensures and delivers all the necessary assets with the optimal quality allowed by the client platform resources. The

Asset Manager will start the application before all the assets are available but continue background downloading the remaining assets as the application is running, all implemented without compromising the optimal quality allowed by the client platform resources.

Solving the Proliferation of Both Hardware Platforms and Software Data Objects:

One difficult task for developing a client side application is reconciling variety of client platforms and variety of data objects. The present invention, through the deployment of the Asset Manager running on each client machine, reconciles beautifully the hardware and software issues. In doing so, the present invention ensures that whatever the resources of each client machine are used fully and efficiently. Importantly, because of its flexibility to accommodate a wide range of platforms, the present invention enables platform-independence beyond that of "Mac. vs. PC".

In particular, one of the benefits of the present invention is that a developer needs only to concentrate on expressing the policies made up of priorities. The detailed implementation of carrying out these policies on a variety of client platforms is automatically done by the Asset Manager. Consequently, from the point of view of a developer, the proliferation of client machine platforms is no longer an obstacle. The programming model and the Asset Manager together ensure the getting of a necessary set of assets downloaded to the client machines at the necessary point in time in order for the application to proceed in a visually appealing and rational manner, and independent of client platforms. Moreover, from the point of view of an end user, the various differences of client platforms are transparent.

To summarize, the programming model offers a standardized way to organize, describe, and record assets. It supports multiple levels of quality for each asset and multiple levels of priority for assignment to each asset tagged with a certain quality level. In doing so, this programming model offers a standardized and convenient way to specify asset policies. The developers record the assets needed by the application on for example a text file called the Manifest. On each client machine, this Manifest is later used by the Asset Manager as the client application runs. The Asset Manager downloads the Manifest, and refers to it to implement the policies specified in the application's codes.

BRIEF DESCRIPTIONS OF THE DRAWINGS

The accompanying drawings, which are incorporated in and form a part of this specification, illustrate embodiments of the present invention and, together with the description, serve to explain the principles of the invention.

Figure 1 illustrates an exemplary computer system used as part of an asset management system in accordance with one embodiment of the present invention.

Figure 2 depicts a generic on-line setup for delivering information, where multiple clients are running browsers to access the information residing on a HTTP (Hyper Text Transfer Protocol) Web server across from the Internet.

Figure 3 depicts a client-server application more sophisticated than the one depicted in Figure 2. It is a chat application being run using the Internet. This illustrates an example of an interactive on-line application.

Figure 4 depicts a generic arrangement for running an interactive on-line application. These client machines are drawn with different sizes to indicate different platforms having different characteristics.

Figures 5A, and 5B are tables summarizing the possible client platform variations organized according to connection bandwidths, CPU speeds, and memory sizes. Figure 5 C gives a board summary of categories that can be used to distinguish various client platforms.

Figure 6 illustrates the preferred embodiment of the present invention, wherein various different client platforms are depicted. Some client machines are depicted as having local persistent storage's.

Figure 7 illustrates one embodiment of the present invention where one generic client machine has multiple local persistent storages. Each of these storage contains a different storage image of one asset.

Figure 8A depicts how assets listed in a Manifest are used in the programming model of the present invention. Figure 8B depicts the information organization of a generic Manifest.

Figure 9 depicts the internal design of the Asset Manager and illustrates how this design provides extensibility of asset data types. Figure 9 also illustrates the concept of the Asset Store and how it enables switching among client side applications.

Figure 10 is the flow chart associated with Figure 9.

Figure 11 is a flow chart of showing how the Asset Store facilitates application switching for the on-line game of Jeopardy.

DETAILED DESCRIPTION

A method and apparatus for management of assets used in running a client side application is described. In the following description, for purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be obvious, however, to one skilled in the art that the present invention may be practiced without these specific details. In other instances, well-known structures and devices are shown in block diagram form in order to avoid obscuring the present invention.

In the following detailed description of the present invention, some of the interchangeable key terms relating to the present invention are defined in the section below to resolve possible ambiguity and to facilitate future reference.

An *asset* is any data type object being used by a client side application. An asset traditionally refers to a media asset such as an image data object, an audio data object, or a text object. But an asset can also be a non-media asset that refers to any data object that needs to be delivered to the client machine for a client side application.

An *Asset Manager* is a system level software component running on a client machine. Each distinct client machine platform has its own type of Asset Manager.

Programming Model is inter alia a methodology of arranging and organizing code with a particular set of APIs for achieving certain specific purposes.

Quality Layers of an Asset refer to different possible versions of delivering an asset. In the case of an image asset, possible quality layers are black and white, and full color.

Priority Level of an Asset is the developer's assessment of the delivery order for an asset in relation to other assets.

Manifest is a text file containing a list of assets needed in running a client side application. This Manifest also records each asset's associated information such as the quality layer and the priority level to which each asset belongs.

Persistent Storage refers to any local persistent storage device of a client machine such as a harddrive or a CD-ROM.

Furthermore, unless specifically stated otherwise as apparent from the following discussions, it is appreciated that throughout the present invention, discussion utilizing terms such as "run", "download", "cache", "store", "switch", "fetch", "instantiate", or the like, refer to the actions and processes of a computer system, or similar electronic computing device. The computer system or similar electronic device manipulates and transforms data represented as physical (electronic) quantities within the computer system's registers and memories into other data similarly represented as physical quantities within the computer system memories into other data similarly represented as physical quantities within the computer system memories or registers or other such information storage, transmission, or display devices.

With reference to Figure 1, portions of the present invention are comprised of the computer-readable and computer executable instructions which reside, for example, in computer system 10 used as a part of a asset managing system in accordance with one embodiment of the present invention. It is appreciated that system 10 of Figure 1 is exemplary only and that the present invention can operate within a number of different computer systems including general-purpose computer systems, embedded computer systems, and stand-alone computer systems specially adapted for client side applications of client-server applications. Computer system 10 includes an address/data bus 12 for conveying digital information between the various components, a central processor unit (CPU) 14 for processing the digital information and instructions, a main memory 16 comprised of volatile random access memory (RAM) for storing the digital information and instructions, a non-volatile read only memory (ROM) 18 for storing information and instructions of a more permanent nature. In addition, computer system 10 may also include a data storage unit 20 (e.g., a magnetic, optical , floppy, or tape drive) for storing vast amounts of data, and an I/O interface 22 for interfacing with peripheral devices (e.g., computer network, modem, mass storage devices, etc.). It should be noted that the software component for performing the asset management process can be stored either in main memory 16, data storage unit 20, or in an external storage device. Devices which may be coupled to computer system 10 include a display device 28 for displaying information to a computer user, an alphanumeric input device 30 (e.g., a keyboard), and a cursor control device 26 (e.g., mouse, trackball, light pen, etc.) for inputting data, selections, updates, etc.

Generic Arrangement for Running an On-line Application:

Client-server applications prevalent using the Internet today are mostly information based. Figure 2 illustrates a generic arrangement for running such simple on-line

application over the Internet. Any of the three clients shown here (204, 206 and 208) can view the information on a Web page 270 by sending a request across the Internet 202 to a HTTP server 200 containing the Web page 270. This HTTP server 200 then sends the Web page 270 to the client that made the request. The on-line application in this case is not interactive in the sense that clients do not interact with each other.

Figure 3 illustrates the setup for running a chat room program, which is an interactive on-line application comprises a server application and a client application. This setup is similar to Figure 2 with the addition of an application server 300 hosting the chat server side application 370. The server 300 handles the logistics of various end users chatting with each other. In contrast with the on-line application of Figure 2, this chat application 370 is an *interactive* on-line application. Notice that the chat application comprises a server side application 370 and client side applications 374, 376, and 378 (i.e., one client side application of the chat application per one client machine).

The chat client applications 304, 306, and 308 run on clients 204, 206, and 208 respectively. Each of these three chat client applications 304, 306, and 308 works with browsers 274, 276, and 278 respectively to fetch assets 322 and 344, and to drive the client user graphic interface.

Figure 4 illustrates a generic setup similar to that of Figure 3 except that all three clients are depicted with different shapes for indicating that these three client machines belong to three *distinct* client platforms. Figure 4 provides a more realistic situation faced by a developer of the client side application for an interactive on-line applications. For a developer of client side interactive on-line applications, the objective is to deliver media rich applications. But the end users can be very far away, making connection bandwidth of the client machine a major consideration in delivering large sized asset such as a streaming

audio file across the Internet. Moreover, once a certain asset has been downloaded to a client machine, the CPU must be powerful enough to handle the sophistication of the asset. Finally, information about an end user's machine is unknown to the developer before the application launches. Because of these three reasons, developing just the client side application of a client-server application is a slow and intricate process for the developers.

Examples of Possible Client Machine Platform Variations:

Figures 5A~ 5C are tables depicting the client platform variability by organizing into three tables some of the possible characteristics for client platforms. These tables shows more explicitly than does Figure 4 in underscoring the platform proliferation problem face by a developer in coding a client side application.

In particular, Figure 5A uses two aspects of platform characteristics: the extrinsic factor of connection bandwidth available to the client machine 501, and the intrinsic factor of memory size of the client machine 502. On the one hand, the bandwidth characteristics is differentiated into High 503 and Low 504. On the other hand, the memory size characteristics is differentiated into Large 505 and Small 506. Altogether, this table 522 presents four possibilities of client platform characteristics (setups): High/Large 511, High/Small 513, Low/Large 512 and Low/Small 514.

In Figure 5 B the connection bandwidth 501 is also used as an aspect of platform characteristics as Figure 5 A. But in Figure 5 B, the aspect of CPU speed 509 is used. The aspect of CPU speed is differentiated into Fast 507 and Slow 508. Altogether, this table 533 presents four possibilities of client platform characteristics (setups): High/Fast 581, High/Slow 583, Low/Fast 582, and Low/Slow 584.

In Figure 5 C, a table lists several more aspects of platform characteristics, including the aspects of connection bandwidth 501, memory size 502, and CPU speed 509. The other aspects are persistent storage types 571 and software functions available on the client machine 572.

In Figure 6, a preferred embodiment of the present invention is illustrated in detail. To take advantage of the programming model offered by the present invention, a developer decides on the types of assets 651–656 that will be used in the client side application of the entire interactive on-line application under design. Also, the developer can offer a gradation of quality layers for each asset. For example, for each image file asset, the developer can easily offer at least two layers of quality: a black and white image file asset as a quality layer, and a full color image file asset as the other quality layer.

In Figure 6, an entire instance of an interactive on-line application (680 and 684 combined) comprises the server side application 680 running on the on-line application server 300, and the client side application 684 running on top of the browser 274 for the client 204. Importantly, the present invention simplifies the development process of the client side application.

In addition to the client side application itself, the developer also supplies the Manifest 697 that records the list of assets and their specific properties. The assets 651–656 used by the client application 684 might be located in one location 200 as shown, or they could be scattered throughout the Internet. Fortunately, because these assets' locations are recorded in the Manifest 697, the application can locate and use these assets by referring to the Manifest 697.

In Figure 6, three clients 204, 206 and 208 are shown across the Internet 202 from an on-line application server 300 and an HTTP server 200. These three clients are drawn with different shapes to indicate three different machine platforms. Firstly, these clients differ in the types of persistent local storages. Client 204 has both a hard disk 664 and a CD-ROM 665, client 206 has only a hard disk 667, and client 208 has no persistent local storage at all. Secondly, Client 206 has a more DRAM than Client 208 does. Finally, Client 206 has low connection bandwidth and Client 208 has high connection bandwidth.

With reference to Figure 6, each client is running its own Asset Manager tailored to the specific strength and weakness of its machine platform. Client 204 is running the Asset Manager 634 that takes advantage of the availability of local storages 664 and 665.

Client 206 is running the Asset Manager 636 that takes advantage of the large DRAM size to minimize repeated download through Client (206)'s low bandwidth connection 696. The Asset Manager 636 downloads the very minimum number of assets having the lowest quality that still allow the application to run in a visually rational manner. And as the application runs, the Asset Manager 636 downloads other assets in the background so that the end user can eventually experience high quality for each asset. Client 208 is running the Asset Manager 638 that takes advantage of the high connection bandwidth 698 and minimizes using the DRAM (the browser cache 648 in this case). This Asset Manager 638 aggressively purges non-immediate assets from the browser cache to continually create room for the needed assets that can be speedily downloaded through the high bandwidth connection 698.

Figure 7 illustrates the concept of multiple storage images of an asset. An asset 777 used by an interactive on-line application (680 and 684) is first stored in certain HTTP server 200. As this asset 777 is downloaded to a client 204 having multiple storages such

as a hard disk 664 and a CD-ROM 665, the asset 777 is stored and retrieved differently depending on the storage type. The asset 777 can be represented by a storage image 755 in the hard disk 665, by a reference to a read-only storage image 744 in the CD-ROM 665, or by a storage image 766 in the browser cache 644. The client's Asset Manager 634 therefore keeps records of how and where an original asset is stored by treating the asset 777 as having different storage images in different storages.

The Programming Model provided by the Present Invention:

Figure 8 A depicts the procedures of making a Manifest 697 by a developer who follows the present invention's programming model. In essence, Figure 8 A provides the schematics of the present invention's programming model for making a Manifest 697. First, the developer specifies all of the assets needed by the application 384. Then for each asset, the developer designs several levels of quality 805. Also, each layer of quality for any type of asset is given a quality tag. For example, if asset A is an image file, then the developer naturally can have at least two layers of quality, i.e., monochrome and full color. After each asset is further differentiated into levels of quality 805, the developer then determine the minimum sets of assets required to run the application at a particular point of running the application. For example, the developer uses priority tags to classify all assets into assets designated with priority 1 (for highest priority), with priority 2 (for the next highest priority), and so on. As a result, in step 808 the developer codes into asset management policies for the logistics of how the application utilizes the assets. Referring again to Figure 6, these policies can be coded in the server side application 680, or in the client side application 684, or spread between both (680 and 684).

In conjunction with coding policies expressed via priority levels, the developer also records the information about asset quality 804 and asset priority level 806 in the Manifest

697. In the final step 877, the Asset Manager 634 refers to the Manifest 697 of the application to interact with the client side application 684.

It should be apparent to one of ordinary skill in the art that the actions of the developer described supra could be performed manually, or automatedly using computers, or partly manually and partly automatedly. The method produces similar results irrespective of method implementation details.

Figure 8 B shows in more detail the information types and formats contained within a Manifest 697. For each asset listed in the Manifest 697, a unique ID number 851 is given. The other parameters of each asset comprise asset data type 852, unique symbolic name 853 for coding the application, asset file size 855, check sum 856, quality layer 857, URL (Uniform Resource Locator) 858 indicating the storage location of the asset, and priority level 859.

The Internal Components of the Asset Manager:

The Asset Manager is a script interpreting engine that supports and exports an interface containing functions for managing the assets needed by the client side application. The client side application invokes the functions of the Asset Manager to manage all of the assets needed to run the application. As the application launches, the Asset Manager object and the Asset Registry object are instantiated. Also the Asset Manager comprises a group of data type factories objects, with one data object type factory per each assets types supported by the Asset Manager. These data type factories are also loaded together with the Asset Manager. For example, some of the common data type factories are the text data type factory for managing text assets, the image data type factory for managing image assets, the

audio data type factory for managing audio assets, and so on. All of these factory types are registered in the Asset registry.

When the application requests a particular asset, e.g., an asset with ID equals to 3, the Asset Manager looks in the Manifest to find that the data type of asset having ID which is equals to 3 is an image data type. Then the Asset Manager calls upon the image data type factory to create an object in response to the request. This asset is stored in the central location (the Asset Store) for storing all objects created by data type factories in response to request.

Specifically, with reference to Figure 9, the internal components of the Asset Manager 634 are discussed. First the client side application of a client-server application, say an interactive on-line application, is coded according to the present invention's programming model. As the client side application (684 of Figure 6) launched, both the Asset Manager object 634 and the asset registry object 920 are instantiated, while the Manifest 697 becomes downloaded from the application server 300. Instantiation of computer objects is well known in the computing arts. The Asset Manager object 634 comprises a set of data object factories 901-905, with typically one data object factory per asset type specified in the Manifest 697. These data object factories 901-905 are registered to the asset registry object 920. As the application runs, whenever an image asset is needed, the image data object factory 902 creates an image asset object 999 in a well known location called the Asset Store 940. The image player 942, the audio player 944, and the client side application check the Asset Store 940 for matching files and object types. For example, the image player 942 would find and play an image asset object 999 inside the Asset Store 940.

With reference to Figure 10, a flow chart is given for outlining the schematics of Figure 9. In the step 1002, as the application launches, the Manifest is loaded to the client. In step 1004, the application notifies the Asset Manager to load the asset with an ID of 3 for example. In step 1006, the Asset Manager searches in the Manifest to see that the asset having ID that is equal to 3 and finds the asset's data type to be the image data type. In the step 1008, the Asset Manager pulls from the asset registry the image data object factory in order to load this particular image data object. In the step 1010, the image data object factory creates an image object representing the image asset in memory. In the step 1012, the image data object factory adds the image object to the asset store.

Extensibility of Asset Types for the Asset Manager:

As discussed above with reference to Figure 9, the Asset Manager supports a group of data object factories, wherein each data object factory responsible for managing a specific type of asset. This particular design of the Asset Manager is a powerful feature of the present invention because the support for managing a new type of assets can be easily added to the Asset Manager without affecting existing codes. As support for any new type of assets is added to the Asset Manager, no change to existing code of the Asset Manager is necessary. To add support to any new type of assets, the mere addition of a new data type factory to the Asset Manager would suffice. Consequently, this design of the Asset Manager enables the extensibility of asset types.

For example, with reference to Figure 9, if a new type of asset that needed to be supported by the Asset Manager is a "movie" asset, the developer simply codes the movie data object factory 905 for inclusion in the group of data object factories for the Asset Manager 634. Then the developer makes available a movie player 946 on the client machine

for playing the movie assets. New code for a new type of asset can be added without affecting any existing codes of the client side application or the Asset Manager.

Another important aspect of the present invention's data type extensibility is the ability to support assets of very large sizes such as a streaming data object, without affecting existing code. For example, if a very long download of a very large size audio asset becomes necessary in running the client side application, the Asset Manager can create an illusion that a download has been completed. For a streaming asset such as an audio asset, the data object representing this audio asset does not have to be fully instantiated in the Asset Store by the audio data type factory. That is to say, partially instantiated asset data object can be accessed by the client side application as if the asset data object were completely instantiated. An audio player can actually begin to play the partially instantiated audio data object while the audio data type factory of the Asset Manager continues to instantiate the rest of the audio data object.

The Asset Store and switching Applications:

The Asset Store plays a facilitating role when an end user is switching between different interactive on-line applications. Specifically, the Asset Store can store all of the instantiated assets shared by various client side applications. Then, any of these client side applications can directly access and use these instantiated assets.

With reference to Figure 11, a flow chart is presented to show how the Asset Store can be applied for playing the game of JEOPARDY (TM) on-line. For example, if a user wants to play the game of JEOPARDY (TM) on-line with some other players, the user first logs on-line to launch the lobby application of JEOPARDY (TM) to meet potential opponents, as outlined in steps 1102, 1104 and 1106. Lobby applications are well known

in the online computer gaming arts. Then, in the step 1108, once the user has decided who the other players are, the game application for Jeopardy is launched in the step 1110. Finally, as a game is finished in the step 1114, the user can go back to the "lobby" to find some other new opponents by launching the lobby application once again, as outlined in sequential steps of 1118, 1122, then 1104. Throughout the whole process, the user would have switched between the lobby application and the game application at least twice, as in steps 1124 and 1122. The Asset Store facilitates the switching between these two applications because it retains all of the assets needed by the lobby application when the user switches to the game application. So when the user switches back to the lobby application from the game application, the assets needed by the lobby application are already in place in the Asset Store, ready to be used. Similarly, because the Asset Store retains all of the assets needed by the game application, as the user switches from the lobby application back to the game application, the game application can start right away without waiting for its assets to get downloaded. As a result, the Asset Store enhances the user's on-line experience of playing JEOPARDY (TM) because the Asset Store facilitates the switching between the lobby application and the game application.

The foregoing descriptions of specific embodiments of the present invention have been presented for purposes of illustration and description. They are not intended to be exhaustive or to limit the invention to the precise forms disclosed, and obviously many modifications and variations are possible in light of the above teaching. The embodiments were chosen and described in order to best explain the principles of the invention and its practical application, to thereby enable others skilled in the art to best utilize the invention and various embodiments with various modification as are suited to the particular use contemplated. The scope of the invention is intended to be defined by the Claims appended hereto and their equivalents.

CLAIMS

What is claimed is:

1. For a client-server system having a plurality of different client platforms, a computer implemented method of generating a client side application of a client-server application, said method comprising:

providing a programming model to be followed by developers for developing said client side application;

providing a programming model to be followed by developers for upgrading said client side application;

generating a file associated with said client side application to record an asset required by said client side application, and wherein at least one of said assets is a data object; and

managing programming logistics for delivering assets required by said client side application, and wherein said managing step is executed on at least one client machine of said client-server system by a platform-specific system level software component of a client machine.

2. The computer implemented method of Claim 1, wherein said programming model requires steps comprising:

specifying multiple layers of quality for at least one of said assets, wherein said asset is a data object having data types selectable from a group comprising video data objects, audio data objects, code data objects, data chunk data objects, graphic data objects, image data objects;

setting a level of priority for delivering at least one of said assets to said client machine;

centralizing in said file information associated with at least one of said assets required by said client application;

designing asset delivering policies with said centralized information of said file; and

providing extensibility of new asset data types, wherein adding new asset data types does not require changing existing code.

3. The computer implemented method of Claim 2, wherein said multiple layers of quality comprise:

a plurality of audio quality layers ranging from low fidelity to high fidelity; and

a plurality of image quality layers ranging from monochrome to full color.

4. The computer implemented method of Claim 2, wherein said priority levels are ranked by positive integers, and an integer one designates an asset having the highest download priority.

5. The computer implemented method of Claim 2, wherein for each said asset, said centralized information in said file comprises:

a field of unique ID number;

a field of data type; and

a field of unique symbolic name.

6. The computer implemented method of Claim 2, wherein said policies need to be coded only once for each client platform, and wherein said policies are coded with said download priorities specified in said file.

7. The computer implemented method of Claim 2, wherein said addition of new asset data types is implemented by adding a data object factory per one new asset data type in

said software component without affecting existing code of said software component and said client side application.

8. The computer implemented method of Claim 1, wherein said managing step further comprises the steps of:

customizing said asset management software component for each type of client platform, wherein said software component manages asset delivery for said client application running on each client machine of said client-server system;

fetching said text file to each said client machine;

prioritizing download ordering of assets with reference to said file;

unifying multiple storage images of each said asset, wherein each of said storage images represents said asset in a distinct persistent memory storage location;

storing in a central memory location all assets shared by a plurality of client applications, wherein said central memory location is termed an Asset Store;

minimizing asset download to avoid interrupting sessions of said client side application session;

9. The computer implemented method of Claim 8, wherein each said type of client platform having characteristics ranges:

from high to low connection bandwidths;

from large to small dynamic memory sizes;

from high to low CPU processing speeds; and

from several to no persistent memory storages.

10. The computer implemented method of Claim 8, wherein said client application can start before all of said required assets are available, and wherein said software component continues background downloading not-yet downloaded assets.

11. The computer implemented method of Claim 8, wherein said multiple storage images are uniquely keyed and encoded in addition to said asset located on a HTTP Web server, and wherein said software component use available persistently stored storage images of said assets to avoid downloading said asset from said HTTP Web server.

12. The computer implemented method of Claim 8, wherein said shared assets minimizes latency in switching between applications.

13. The computer implemented method of Claim 8, wherein said minimizing step comprises:

searching said text file for all assets having priority levels lower than assets having the highest priority level;

downloading said assets having a highest priority level;

starting said client application as soon as said assets having the highest priority level are completely downloaded;

downloading said lower priority assets in the background while said client application is in session; and

downloading said lower priority assets while said client application is in between sessions.

14. For a client-server system having a plurality of client machines from different client platforms, a computer-readable medium having stored thereon instructions for causing said client machines to deliver assets required by a client side application of a client-server application, comprising code for supporting:

a plurality of assets fetched from a plurality of servers;

a server component generating a file to list said required assets and to specify properties associated with said assets;

a management component for delivering said assets in accordance to said plurality of policies; and

an extensible set of data object factories, wherein each of said data object factories is associated with a distinct asset data type.

15. The computer-readable medium of Claim 14, wherein each said asset is a data object selectable from a group comprising video data objects, audio data objects, code data objects, data chunk data objects, graphic data objects, and image data objects.

16. The computer-readable medium of Claim 14, wherein said file is a manifest associated with said client side application, said manifest comprising:

a field of unique ID number for each said asset;

a field of data type for each said asset; and

a field of unique symbolic name for each said asset.

17. The computer-readable medium of Claim 14, wherein the priority levels assignable to each of said assets are designated by positive integers, wherein an integer one designates an asset with the highest download priority level.

18. The computer-readable medium of Claim 14, wherein for at least one of said client machines, said policies specify delivering a necessary platform-specific subset of said assets at a necessary point in time in order for said client side application to proceed in a visually appealing and rational manner.

19. The computer-readable medium of Claim 14, wherein said management component downloads said assets to at least one of said client machines according to said policies, and wherein said management component fetches said assets directly from a plurality of local persistent memory locations, provided said assets have been downloaded and stored in said plurality of local persistent memory locations.

20. The computer-readable medium of Claim 14, wherein at least one of said data object factories, in the dynamic memory of said client machines, instantiates a data object representing an asset with the same data type as said data object.

21. The computer-readable medium of Claim 20, wherein at least one of said instantiated data object is stored in a common dynamic memory location, wherein another client side application requiring such said data object can directly access said data object without any further data object instantiation.

22. The computer-readable medium of Claim 14, wherein said servers are HTTP Web servers.

23. The computer-readable medium of Claim 14, further comprising a plurality of priority levels assigned to said assets.

24. The computer-readable medium of Claim 14, further comprising a plurality of policies for delivering said assets to said client machines.

25. The computer implemented method of Claim 2, wherein for each said asset, said centralized information in said file further comprises:

a field of URL;

a field of quality;

a field of checksum; and

a field of download priority level.

26. The computer-readable medium of Claim 14, wherein said file is a manifest associated with said client side application, said manifest further comprising:

a field of URL for each said asset;

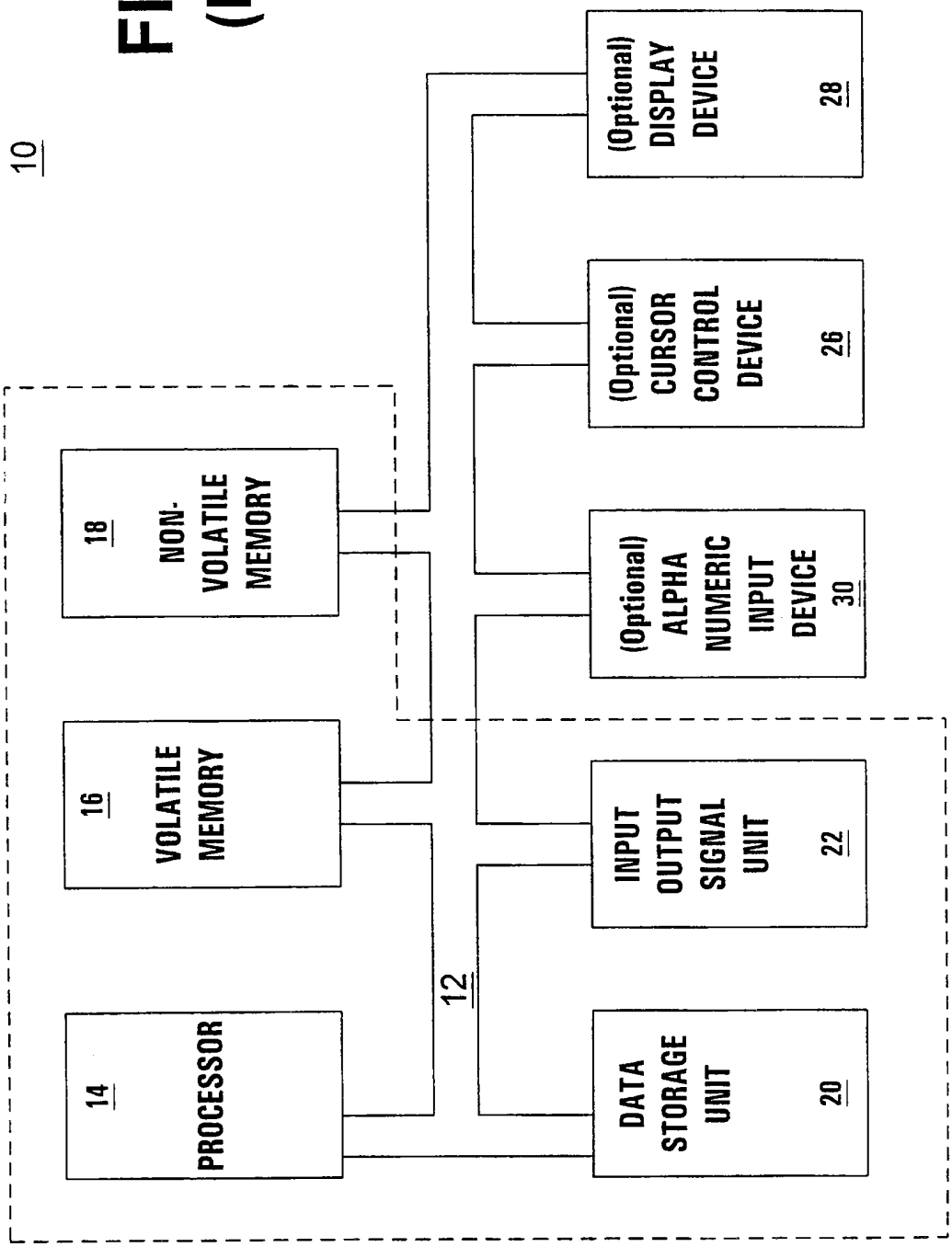
a field of asset file size for each said asset;

a field of quality for each said asset;

a field of checksum for each said asset; and

a field of download priority level for each said asset.

FIGURE 1
(Prior Art)



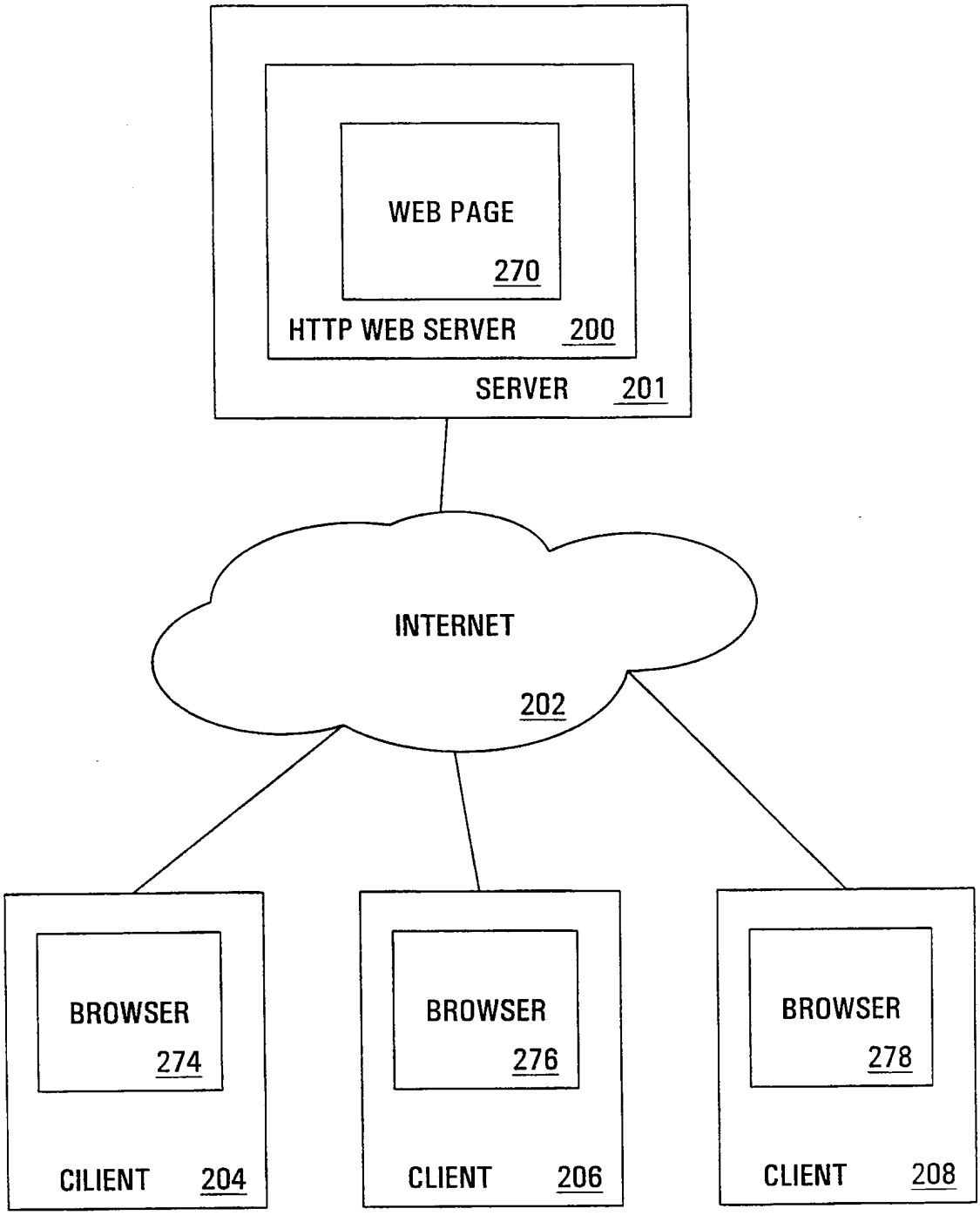


FIGURE 2
(Prior Art)

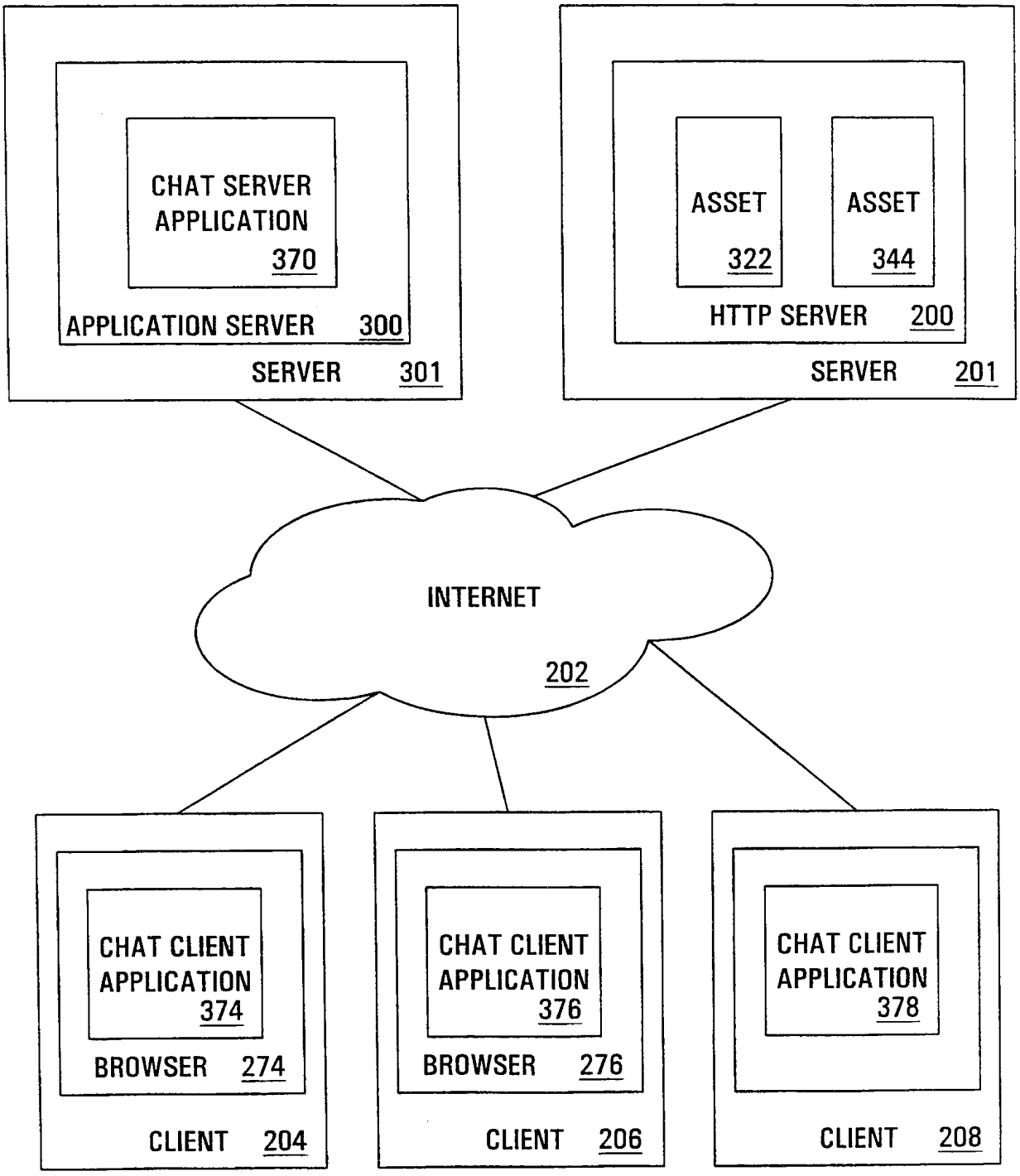


FIGURE 3
(Prior Art)

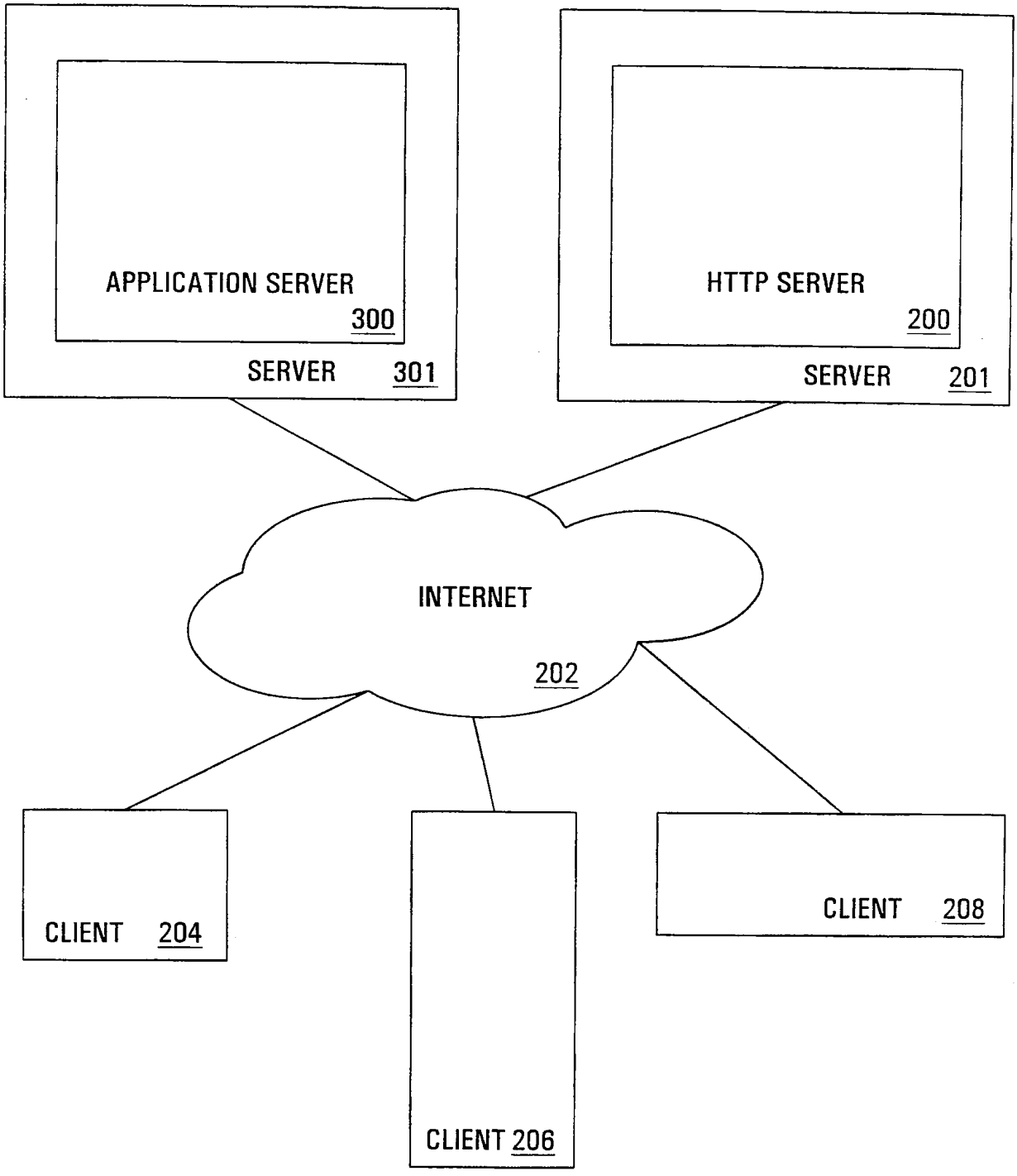


FIGURE 4

MEMORY SIZE 502 / BANDWIDTH 501	LARGE 505	SMALL 506
HIGH 503	HIGH/LARGE 511	HIGH/SMALL 513
LOW 504	LOW/LARGE 512	LOW/SMALL 514

FIGURE 5A

CPU SPEED 509 / BANDWIDTH 501	FAST 507	SLOW 508
HIGH 503	HIGH/FAST 581	HIGH/SLOW 583
LOW 504	LOW/FAST 582	LOW/SLOW 584

FIGURE 5B

CLIENT MACHINE CHARACTERISTICS	RANGE OF VARIATION
CONNECTION BANDWIDTH 501	HIGH ⟷ LOW
MEMORY SIZE 502	LARGE ⟷ SMALL
CPU SPEED 509	FAST ⟷ SLOW
PERSISTENT STORAGE 523	{ HD, CD-ROM } ⟷ NONE
AVAILABLE THIRD PARTY APPLICATION	{ AUDIO PLAYER, VIDEO PLAYER } ⟷ NONE

FIGURE 5C

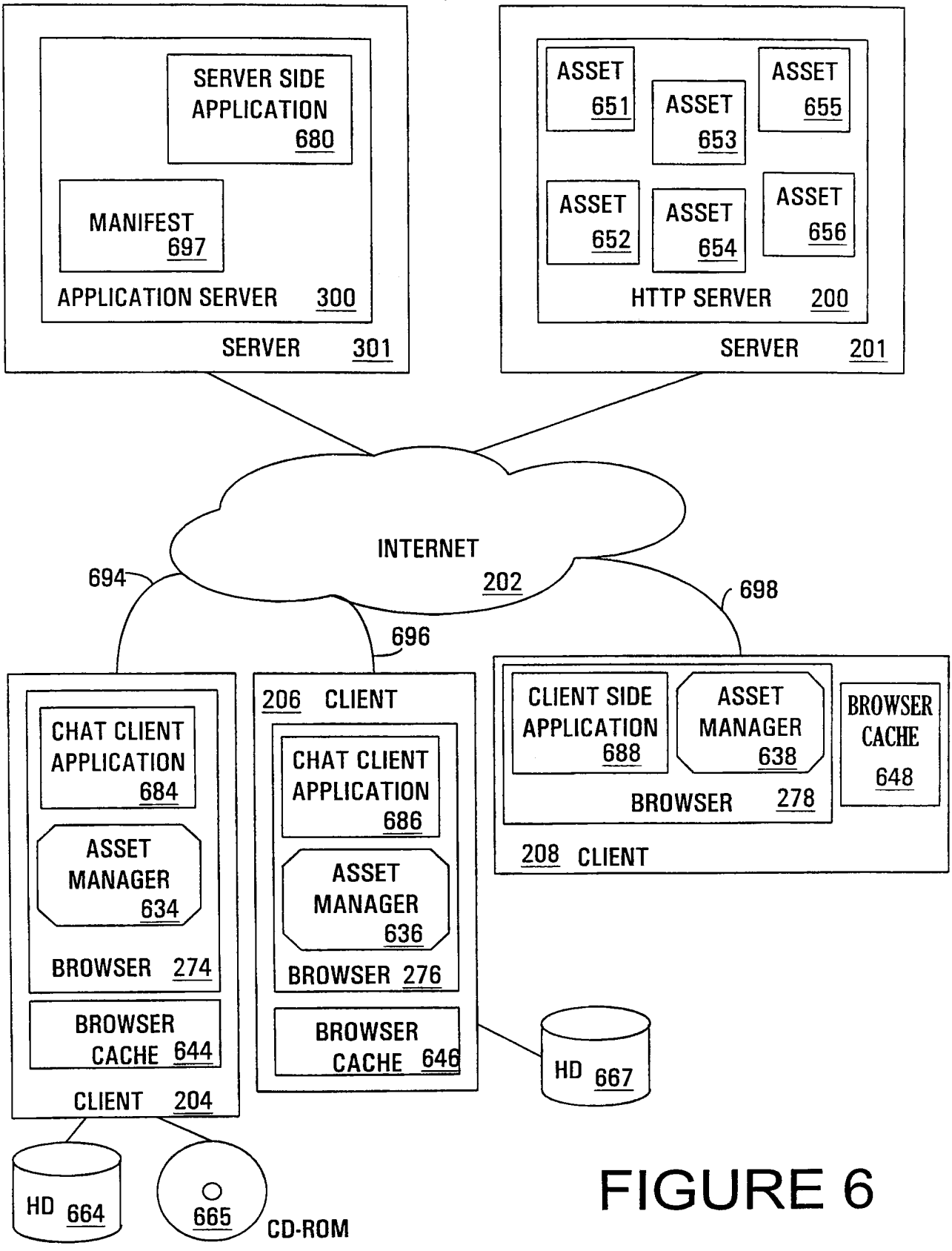


FIGURE 6

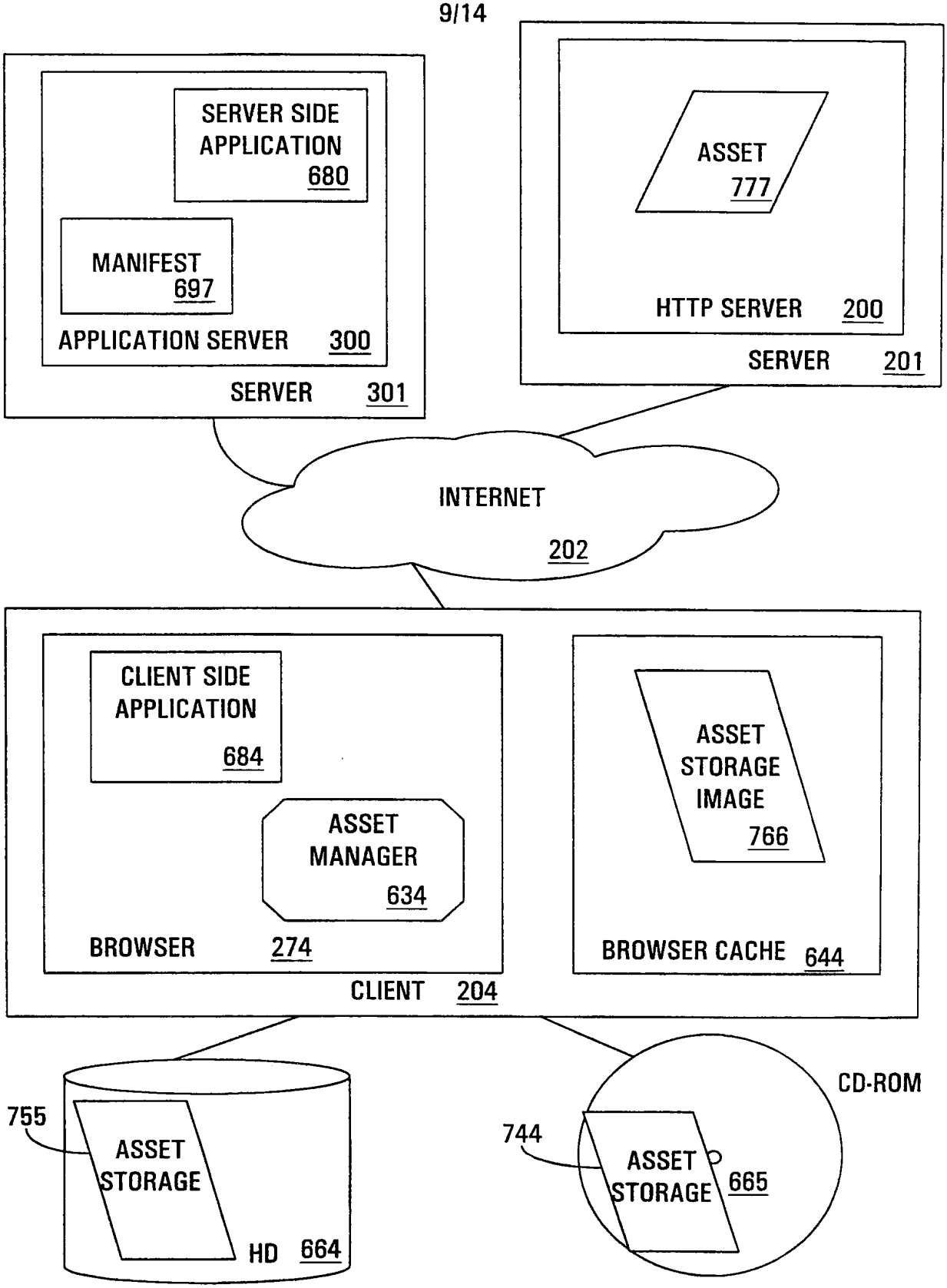


FIGURE 7

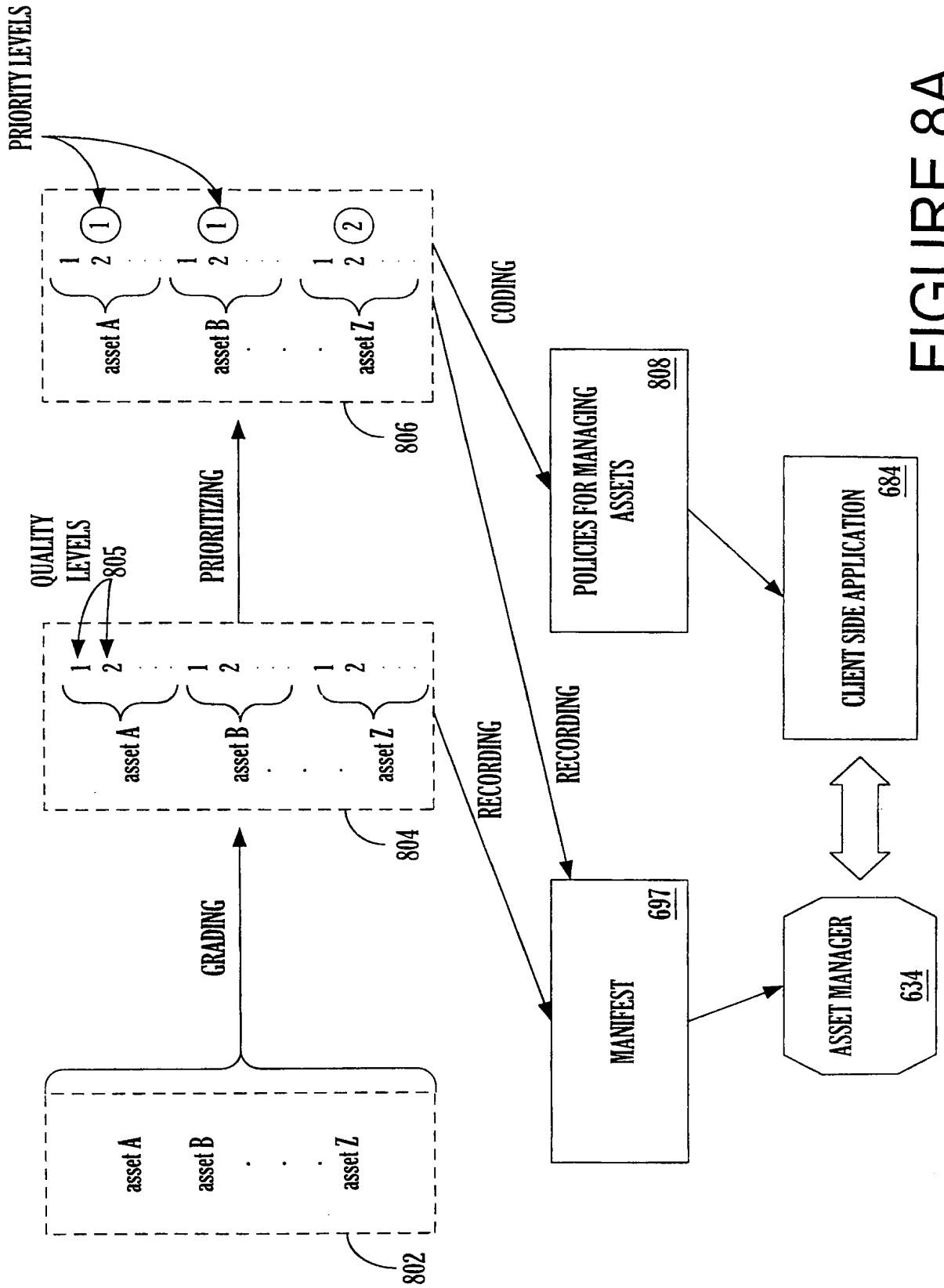


FIGURE 8A

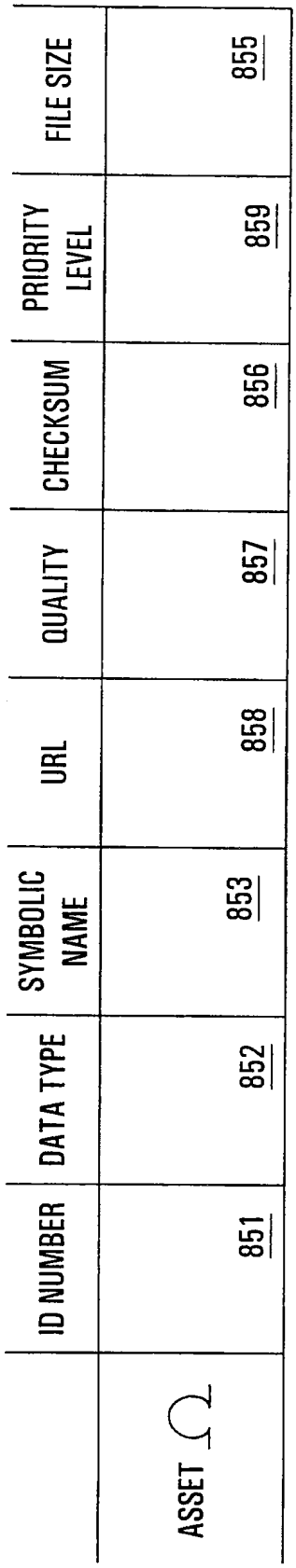


FIGURE 8B

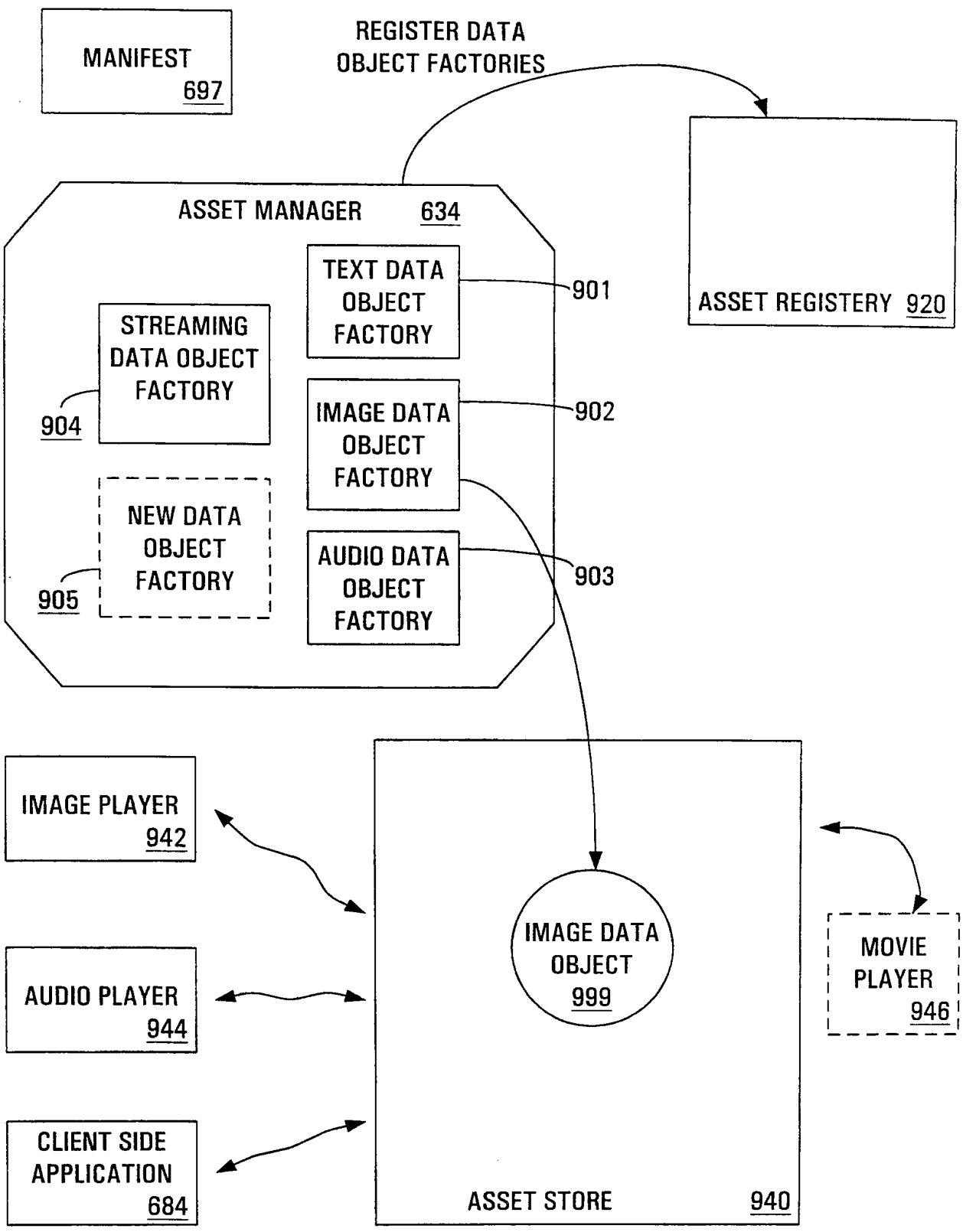


FIGURE 9

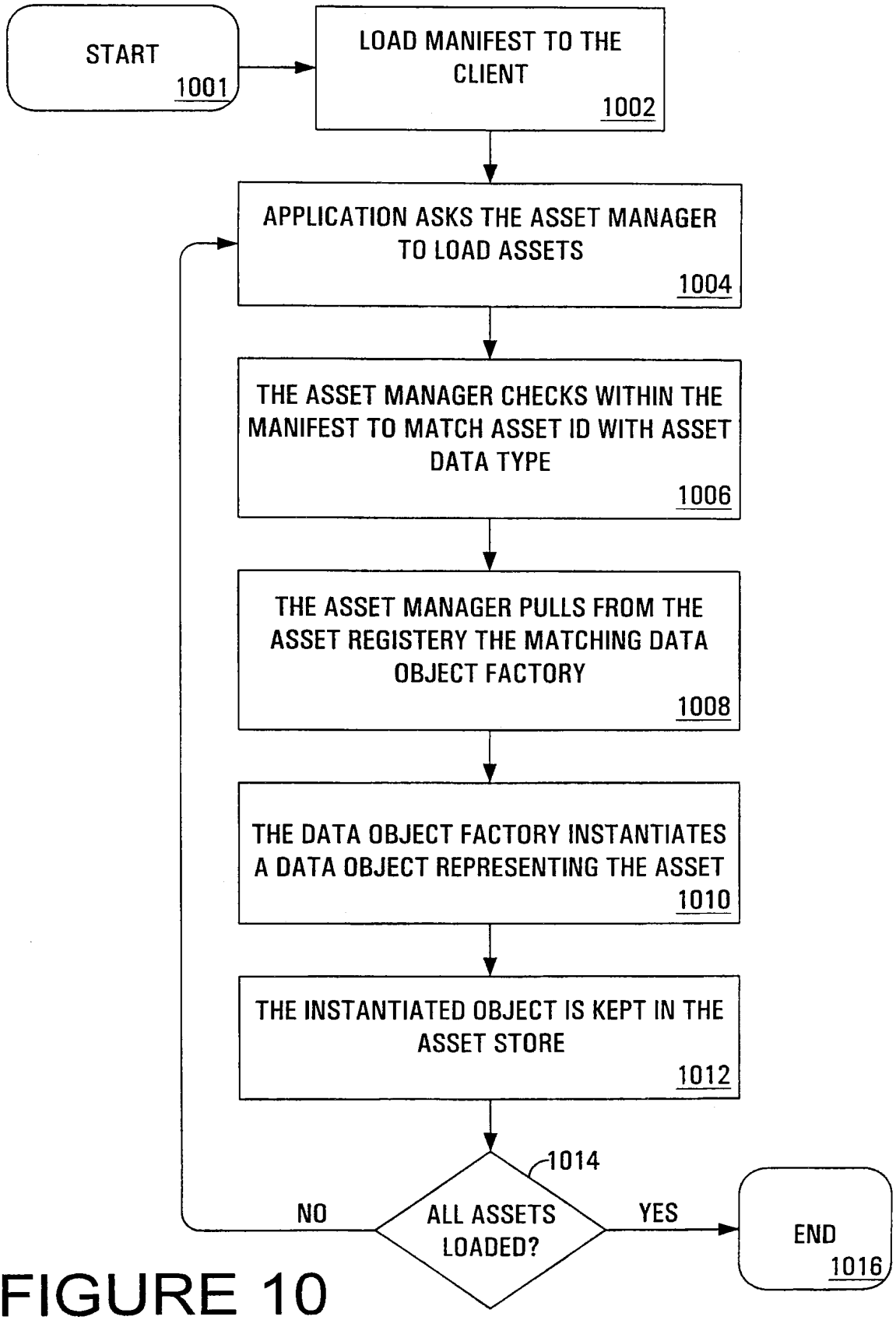


FIGURE 10

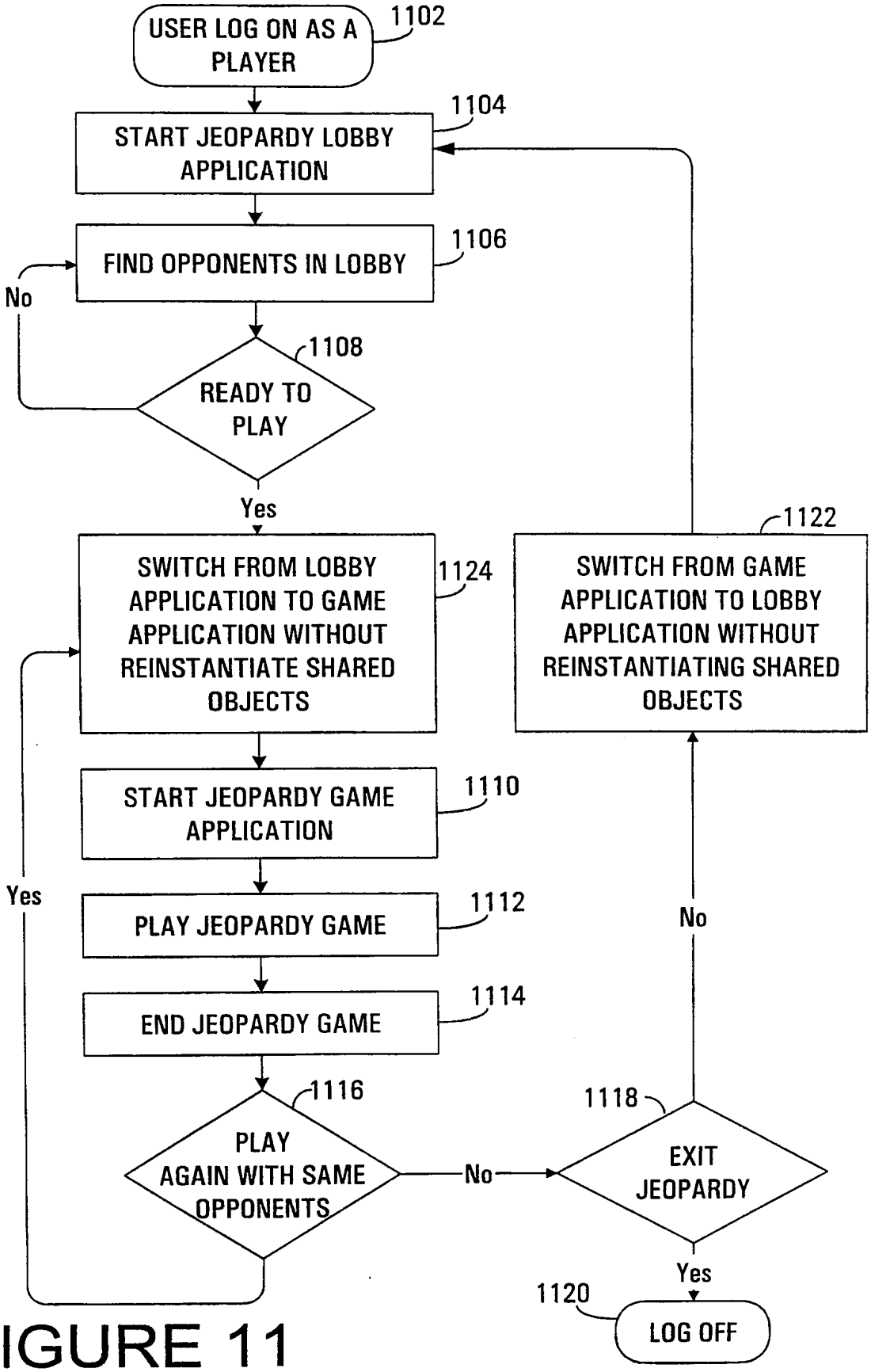


FIGURE 11

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 00/03346

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F9/44

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

EPO-Internal, INSPEC, IBM-TDB

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5 815 711 A (SAKAMOTO KAZUHIKO ET AL) 29 September 1998 (1998-09-29) abstract column 1, line 1 -column 2, line 11 column 2, line 55 -column 4, line 67 claim 1 ---	1,14
A	NOBORU AKIMA ET AL: "INDUSTRIALIZING SOFTWARE DEVELOPMENT: A JAPANESE APPROACH" IEEE SOFTWARE,US,IEEE COMPUTER SOCIETY. LOS ALAMITOS, vol. 6, no. 2, 1 March 1989 (1989-03-01), pages 13-21, XP000007936 ISSN: 0740-7459 the whole document --- -/--	1,14

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
11 July 2000	18/07/2000

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31–70) 340–2040, Tx. 31 651 epo nl, Fax: (+31–70) 340–3016	Beltrán-Escavy, J

1

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 00/03346

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
A	US 5 655 091 A (DRAKO DEAN M ET AL) 5 August 1997 (1997-08-05) abstract column 1, line 1 -column 2, line 55 column 3, line 40 -column 4, line 30 column 6, line 3 -column 6, line 32 claim 1	1-3,8, 11,14, 15,22
A	EP 0 649 121 A (IBM) 19 April 1995 (1995-04-19) abstract page 3, line 1 -page 6, line 43 page 7, line 54 -page 8, line 25 claims 6,7	1,8,11, 14,15,22
A	WO 97 15018 A (BELL COMMUNICATIONS RES) 24 April 1997 (1997-04-24) page 1, line 1 -page 5, line 3 claim 1	1,8,11, 14,15,22

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/US 00/03346

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5815711 A	29-09-1998	JP 8137678 A	31-05-1996
US 5655091 A	05-08-1997	US 5446866 A	29-08-1995
		DE 69216379 D	13-02-1997
		EP 0553549 A	04-08-1993
		JP 6059989 A	04-03-1994
EP 0649121 A	19-04-1995	AT 189074 T	15-02-2000
		DE 69422679 D	24-02-2000
		DE 69422679 T	06-07-2000
		JP 7175868 A	14-07-1995
		JP 10207965 A	07-08-1998
		KR 143358 B	17-08-1998
		US 5734719 A	31-03-1998
WO 9715018 A	24-04-1997	NONE	